
archetypal Documentation

Release 2.7

Samuel Letellier-Duchesne

Apr 19, 2022

GETTING STARTED

1	Description	3
2	Indices and tables	199
	Bibliography	201
	Index	203

archetypal is a Python package designed with the objective of helping building energy modelers and researchers maintain collections of building archetypes. *archetypal* depends on [eppy](#) for EnergyPlus models and makes use of great packages such as [pandas](#) for data structure processing and [tsam](#) for time series aggregation.

DESCRIPTION

As building energy modelers ourselves, we found it was sometimes difficult to use scripting language to retrieve, modify, simulate and analyze Building Energy Models (BEM). This is why *archetypal* was created. We built two main capabilities into the package:

1. The conversion of EnergyPlus to [TRNBuild](#) models (shout out to TRNSYS users!)
2. The conversion of EnergyPlus to [UMI](#) Template Files.

archetypal also features a [Command Line Interface \(CLI\)](#) which means that users can execute commands in the terminal instead of writing a python script. In addition, we believe reproducible research through Jupyter Notebooks, for instance, is the way forward. Therefore, all the modules are discoverable and can be imported independently.

1.1 Installation

1.1.1 Requirements

Warning: Prior to installing this package, you must have EnergyPlus version 9.2.0 ([download](#) here at the bottom of the page).

[EnergyPlus](#) should be installed in its default location. On Windows that would be in *C:\EnergyPlusV9-2-0* and on MacOS that would be in */Applications/EnergyPlus-9-2-0*.

It is also recommended that the older transition programs be installed as well. These programs allow older IDF files (versions 7.2 and below) to be upgraded to version 9-2-0. Since these, don't come by default with EnergyPlus, they need to be installed by hand. A script has been created for windows (see [Installation from scratch](#)). For macOS, refer to the [supplementary conversion programs](#).

1.1.2 Installation from scratch

This first step should be helpful for users that are not familiar with python environments. If you already have python installed and think that you can manage the installation a new package using *pip*, then you can skip to the next section.

Download & Install MiniConda (or the full Anaconda)

found at the following URL: <https://docs.conda.io/en/latest/miniconda.html>

Launch the executable and select the following settings:

- InstallationType=JustMe
- AddToPath=Yes (there might be a warning, but ignore it)
- RegisterPython=Yes
- Installation path=%UserProfile%\Miniconda3

Check if everything is ok by running `conda list` in the command line (make sure to open a new command line window just in case). You should see something like this:

```
C:\Users\archetypal>conda list
# packages in environment at C:\ProgramData\Miniconda3:
#
# Name                          Version                      Build  Channel
asn1crypto                      1.2.0                      py37_0
ca-certificates                2019.10.16                  0
certifi                         2019.9.11                  py37_0
...
win_inet_pton                  1.1.0                      py37_0
wincertstore                   0.2                        py37_0
yaml                           0.1.7                      hc54c509_2
```

Install EnergyPlus & Conversion Programs

EnergyPlus is a prerequisite of archetypal. It must be installed beforehand. Moreover, archetypal contains routines that may download IDF components that are coded in earlier versions of EnergyPlus (e.g., 7.1). For this reason, users should also download the [supplementary conversion programs](#), and install the content in the EnergyPlus installation folder:

- On Windows: `C:\EnergyPlusV9-2-0\PreProcess\IDFVersionUpdater` (For Windows, see automated procedure below).
- On MacOS: `/Applications/EnergyPlus-9-2-0/PreProcess/IDFVersionUpdater`

On Windows, this installation procedure can be automated with the following [script](#) which will download and install EnergyPlus as well as the supplementary conversion programs.

To use the script, follow the next steps. First git must be installed beforehand with default installation parameters. See <https://git-scm.com/downloads> to download git. Then the following commands will change the current directory to the user's Downloads folder. Then the script will be downloaded using the `git clone` command. Finally the script will be executed. Copy the whole code block below in Command Prompt and Hit Enter:

```
cd %USERPROFILE%\Downloads
git clone https://gist.github.com/aef233396167e0f961df3d62a193573e.git
cd aef233396167e0f961df3d62a193573e
install_eplus_script.cmd
```

To install *archetypal*, follow the steps detailed below in [Installing using pip](#)

1.1.3 Installing using pip

If you have Python 3 already installed on your machine and don't bother to create a virtual environment (which is highly recommended), then simply install using the following command in the terminal:

```
pip install archetypal
```

Hint: If you encounter an issue during the installation of archetypal using pip, you can try out [Installing using conda \(Anaconda\)](#) instead.

1.1.4 Installation within a Virtual Environment

It is highly recommended to use/install *archetypal* on a fresh python virtual environment. If you have any trouble with the installation above, try installing archetypal in a new, clean [virtual environment](#) using venv or conda. Note that this package was tested with python 3.6:

```
python3 -m venv archetypal
source archetypal/bin/activate
```

Activating the virtual environment will change your shell's prompt to show what virtual environment you're using, and modify the environment so that running python will get you that particular version and installation of Python. For example:

```
$ source archetypal/bin/activate
(archetypal) $ python
Python 3.5.1 (default, May  6 2016, 10:59:36)
...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
 '~/envs/archetypal/lib/python3.5/site-packages']
>>>
```

Then you can install archetypal in this freshly created environment:

```
pip install archetypal
```

To use the new environment inside a [jupyter notebook](#), we recommend using the steps described by [Angelo Basile](#):

```
source archetypal/bin/activate
pip install ipykernel
ipython kernel install --user --name=archetypal
```

Next time you [start a jupyter notebook](#), you will have the option to choose the *kernel* corresponding to your project, *archetypal* in this case.

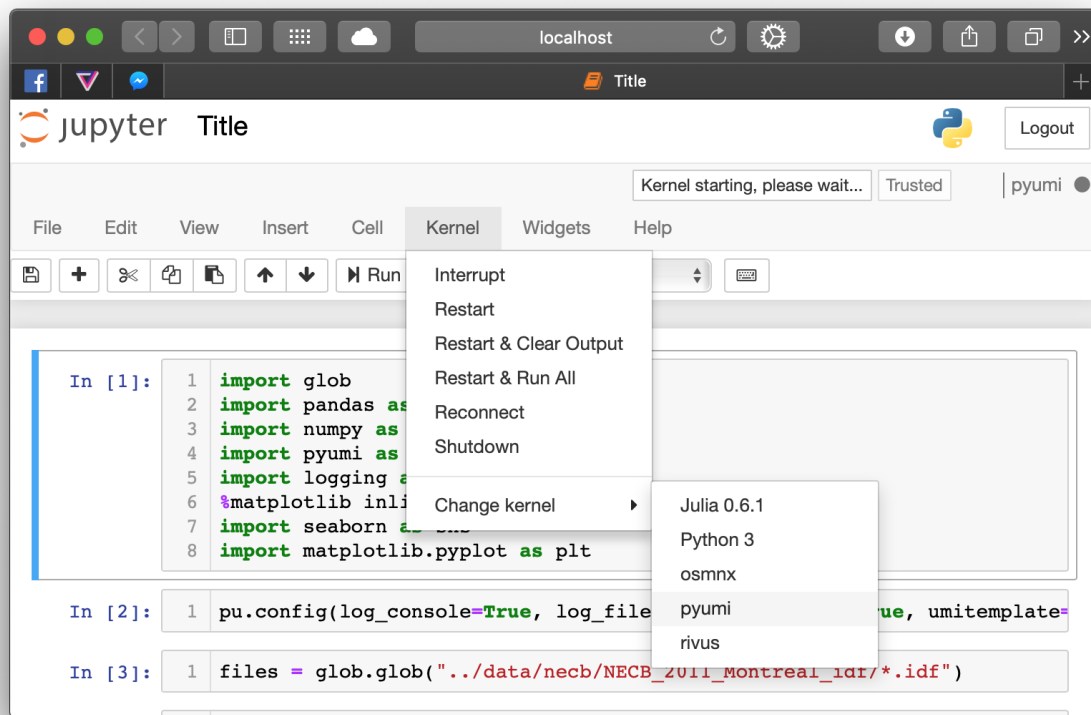


Fig. 1.1: choosing the correct kernel in a jupyter notebook. In the *kernel* menu, select *Change Kernel* and select the appropriate virtual env created earlier (*archetypal* in this case).

1.1.5 Installing using conda (Anaconda)

Hint: If you encounter package dependency errors while installing *archetypal* using pip, you can use conda instead.

Installing with conda is similar to pip. The following workflow creates a new virtual environment (named *archetypal*) which contains the required dependencies. It then installs the package using pip. You will need to download the *environment.yml* file from the github repository. For the following code to work, first change the working directory to the location of the downloaded *environment.yml* file. Here we use the *conda env update* method which will work well to create a new environment using a specific dependency file in one line of code:

```
conda update -n base conda
conda env update -n archetypal -f environment.yml
conda activate archetypal
pip install archetypal
```

1.2 For MacOS/Linux users

MacOS or Linux users must install *Wine* before running *archetypal*. This software will allow MacOS/Linux users to run Windows application (e.g. *trnsidf.exe*).

1.2.1 Wine installation

1. In the Terminal, you have to install Homebrew with the following command line:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

You will have to confirm this action by pressing enter. The Terminal might ask your password, then you have to enter the Admin password (followed by *Enter*:). The installation of Homebrew should take few seconds or minutes.

2. After installing Homebrew, you have to run ‘brew doctor’ (the Terminal might ask you) with the following command line:

```
brew doctor
```

This action will make Homebrew inspected your system to make sure the installation is correctly set up

3. Then you will need to install Xquartz using Homebrew by typing the following command line:

```
brew cask install xquartz
```

4. Finally you can install Wine by copying the following command line:

```
brew install wine
```

For more information about Wine installation, you can visit the following website: <https://www.davidbaumgold.com/tutorials/wine-mac/>

1.2.2 Using WINE with archetypal convert command

The IDF to BUI converter uses an executable installed with TRNSYS (which is Windows only). Users that have bought TRNSYS can copy the trnsidf.exe executable to their UNIX machine (MacOs or Linux) and invoke the *archetypal convert* command with the `--trnsidf_exe` option.

Example:

```
archetypal convert --trnsidf_exe "<path to executable on UNIX machine>" "<path to IDF_
↪file>"
```

You can find the executable trnsidf.exe in the TRNSYS default installation folder: `C:\TRNSYS18\Building\trnsIDF`

1.3 Caching

Archetypal features a caching api aimed at accelerating reproducible workflows using EnergyPlus simulations by reducing unnecessary calls to the EnergyPlus executable or transitioning programs. Concretely, caching an IDF model means that, for instance, if an older version model (less than 9.2) is ran, archetypal will transition a copy of that file to version 9.2 (making a copy beforehand) and run the simulation with the matching EnergyPlus executable. The next time the `IDF()` constructor is called, the cached (transitioned) file will be readily available and used; This helps to save time especially with reproducible workflows since transitioning files can take a while to complete.

As for simulation results, after `archetypal.idfclass.idf.IDF.simulate()` is called, the EnergyPlus outputs (.csv, sqlite, mtd, .mdd, etc.) are cached in a folder structure than is identified according to the simulation parameters; those parameters include the content of the IDF file itself (if the file has changed, a new simulation is required), whether an annual or design day simulation is executed, etc. This means that if `simulate` is called a second time (let us say after restarting a Jupyter Notebook kernel), the `simulate` will bypass the EnergyPlus executable and retrieve the cached simulation results instead. This has two advantages, the first one being a quicker workflow and the second one making sure that whatever `IDF.simulation_files` returns fits the parameters used with the executable. Let us use this in a real world example. First, caching is enabled using the *config* method:

1.3.1 Enabling caching

Caching is enabled by passing the `use_cache=True` attribute to the `archetypal.utils.config()` method. The configuration of archetypal settings are not persistent and must be called whenever a python session is started. It is recommended to put the *config* method at the beginning of a script or in the first cells of a Jupyter Notebook (after the import statements).

```
from archetypal import IDF, config, EnergyPlusVersion
config(use_cache=True, log_console=True)
```

1.3.2 Example

In a Jupyter Notebook, one would typically do the following:

```
idf = IDF.from_example_files(
    "AdultEducationCenter.idf",
    epw="USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw",
    design_day=True,
    annual=False,
    expandobjects=True,
```

(continues on next page)

(continued from previous page)

```
prep_outputs=True,
)
```

If the file would be an older version, archetypal is going to transition the file to EnergyPlus 9.2 (or any other version specified with the `as_version` parameter) and execute EnergyPlus for the *design_day* only.

The command bellow yields a list of output files. These will be located inside a cache folder specified by the `settings.cache_folder` variable (this folder path can be changed using the config method).

```
>>> idf.simulate().simulation_files
[Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳AdultEducationCenter.idf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.audit'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.bnd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.dxf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.eio'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.end'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.err'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.eso'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.expidf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.mdd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.mtd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.mtr'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.rdd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.shd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431bout.sql'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431btbl.csv'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\
↳b07dbcb49b54298c5f64fe5ee730431btbl.htm'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\runargs.
↳json'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\sqlite.
↳err'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\b07dbcb49b54298c5f64fe5ee730431b\\USA_IL_
↳Chicago-OHare.Intl.AP.725300_TMY3.epw')]]
```

Now, if the command above is modified with `annual=True` and set `design_day=False`, then `idf.simulate().simulation_files` should return the annual simulation results (which do not exist yet).

```
>>> idf.simulate(annual=True, design_day=False).simulation_files
```

Now, since the original IDF file (the version 8.9 one) has not changed, archetypal is going to look for the transitioned file that resides in the cache folder and use that one instead of retransitioning the original file a second time. On the other hand, since the parameters of `simulate()` have changed (annual instead of `design_day`), it is going to execute EnergyPlus using the annual method and return the annual results (see that the second-level folder id has changed from `b07dbcb49b54298c5f64fe5ee730431b` to `1cc202748b6c3c2431d203ce90e4d081`; *these ids may be different on your computer*):

```
[Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.audit'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.bnd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.dxf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.eio'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.end'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.err'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.eso'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.expidf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.mdd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.mtd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.mtr'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.rdd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.shd'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081out.sql'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081tbl.csv'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ 1cc202748b6c3c2431d203ce90e4d081tbl.htm'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\
↳ AdultEducationCenter.idf'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\runargs.
↳ json'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\sqlite.
↳ err'),
Path('cache\\b0b749f1c11f28b3d24d1d8978d1140e\\1cc202748b6c3c2431d203ce90e4d081\\USA_IL_
↳ Chicago-OHare.Intl.AP.725300_TMY3.epw')]
```

If we were to rerun the first code block (annual simulation) then it would return the cached results instantly from the cache:

```
Successfully parsed cached idf run in 0.00 seconds
```

Profiling this simple script shows an 8x speedup.

1.4 Converting IDF models

EnergyPlus models can be converted to umi template library files using *archetypal*.

1.4.1 Converting IDF to UMI

The IDF to UMI converter generates an Umi Template from one or more EnergyPlus models (IDF files). The conversion is performed by simplifying a multi-zone and geometric model to a 2-zone and non-geometric template. In other words, a complex EnergyPlus model is converted to a generalized core- and perimeter-zone with aggregated performances.

Conversion can be achieved either with the command line or within a python console (interactive shell). The command line is useful for getting started quickly but does not offer any intermediate like the interactive shell does. If you would rather use archetypal inside a python script, then the archetypal module is fully accessible and documented.

Using the Command Line

Hint: In this tutorial, we will be using an IDF model from the ExampleFiles folder located inside the EnergyPlus folder.

Terminal and Command Prompt may not be the most convenient tool to use, which is quite understandable, since users may be more familiar with graphical interfaces. *archetypal* does not feature a graphical interface as it is meant to be used in a scripting environment.

The first step would be to change the current directory to the one where the idf file is located. When *archetypal* is executed, temporary folders may be created to enable the conversion process. It is recommended to change the current directory of the terminal window to any working directory of your choice.

```
cd "/path/to/directory"
```

An idf file can be converted to an umi template using the *reduce* command. For example, the following code will convert the model *AdultEducationCenter.idf* to a json file named *myumitemplate.json*. Both absolute and relative paths can be used.

```
archetypal reduce "/Applications/EnergyPlus-9-2-0/ExampleFiles/BasicFiles/  
↳AdultEducationCenter.idf" "./converted/myumitemplate.json"
```

Using the Python Console

archetypal methods are accessible by importing the package.

1. Load the file

First, load the EnergyPlus idf file using the *archetypal.idfclass.idf.IDF* class. In the following example, the *AdultEducationCenter.idf* model is used.

```
>>> from archetypal import get_eplus_dirs, IDF
>>> eplus_dir = get_eplus_dirs("9-2-0") # Getting EnergyPlus install path
>>> eplus_file = eplus_dir / "ExampleFiles" / "BasicsFiles" / "AdultEducationCenter.idf"
↳ # Model path
>>> weather = eplus_dir / "WeatherData" / "USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw"
↳ # Weather file path
>>> idf = IDF(idfname=eplus_file, epw=weather) # IDF load
```

2. Create a BuildingTemplate Object

```
>>> from archetypal import BuildingTemplate
>>> template_obj = BuildingTemplate.from_idf(
>>>     idf, DataSource=idf.name
>>> )
```

3. Create an UmiTemplateLibrary Object and Save

```
>>> from archetypal import UmiTemplateLibrary
>>> template_json = UmiTemplateLibrary(
>>>     name="my_umi_template",
>>>     BuildingTemplates=[template_obj]
>>> ).to_dict()
```

1.5 Reading and running IDF files

archetypal is packed up with some built-in workflows to read, edit and run EnergyPlus files.

1.5.1 Reading

To read an IDF file, simply call *IDF* with the path name. For example:

```
>>> from archetypal import get_eplus_dirs, IDF
>>> eplus_dir = get_eplus_dirs("9-2-0") # Getting EnergyPlus install path
>>> eplus_file = eplus_dir / "ExampleFiles" / "BasicsFiles" / "AdultEducationCenter.idf"
↳ # Model path
>>> idf = IDF(eplus_file) # IDF load
```

You can optionally pass the weather file path as well:

```
>>> weather = eplus_dir / "WeatherData" / "USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw"
↳ # Weather file path
>>> idf = IDF(eplus_file, epw=weather) # IDF load
```

1.5.2 Editing

Editing IDF files is based on the eppy package. The *IDF* object exposes the EnergyPlus objects that make up the IDF file. These objects can be edited and new objects can be created. See the [eppy documentation](#) for more information on how to edit IDF files.

Hint: Pre-sets

EnergyPlus outputs can be quickly defined using the *Outputs* class. This class and its methods handle adding predefined or custom outputs to an IDF model. An *Outputs* is instantiated by default in an *IDF* model. It accessed with the `outputs` attribute. For example, the *idf* object created above can be modified by adding a basic set of outputs:

```
>>> idf.add_basics().apply()
```

One can specify custom outputs by calling `add_custom()` with a list of dict of the form `fieldname:value` and then `apply()`. These outputs will be appended to the IDF model only if `apply()` is called. See *Outputs* for more details on all possible methods.

1.5.3 Running

To run an *IDF* model, simply call the `simulate()` function on the *IDF* object. In both cases, users can also specify run options as well as output options.

For the same *IDF* object above:

```
>>> idf.simulate(weather_file=weather)
```

Hint: Caching system.

When running EnergyPlus simulations, a caching system can be activated to reduce the number of calls to the EnergyPlus executable or to reduce time spent on I/O operations such as in `sql` and `htm()` which parse the simulation results. This caching system will save simulation results in a folder identified by a unique identifier. This identifier is based on the content of the IDF file, as well as EnergyPlus simulate options. This system works by invalidating any dependant attributes when independent attributes change.

The caching system is activated by calling the `archetypal.utils.config()` method (or by setting `settings.use_cache = True`), which can also be used to set a series of package-wide options. `config` would typically be put at the top of a python script:

```
>>> from archetypal import config
>>> config(use_cache=True)
```

1.6 Running multiple files

Running multiple IDF files is easily achieved by using the `parallel_process()` method.

Hint: The `parallel_process()` method works with any method. You can use it to parallelize other functions in your script.

To create a parallel run, first import the usual package methods and configure *archetypal* to use caching and to show logs in the console.

```
>>> from path import Path
>>> from archetypal import IDF, config, settings, parallel_process
>>> import pandas as pd
>>> config(use_cache=True, log_console=True)
```

Then, use

```
>>> from archetypal import IDF, config, settings
>>> from archetypal import parallel_process
>>> import pandas as pd
>>> config(use_cache=True, log_console=True)
```

Then, use `glob` to make a list of NECB idf files in the `input_data` directory (relative to this package). The weather file path is also created:

```
>>> necb_basedir = Path("tests/input_data/necb")
>>> files = necb_basedir.glob("*.idf")
>>> epw = Path("data/CAN_PQ_Montreal.Intl.AP.716270_CWEC.epw")
```

For good measure, load the files in a DataFrame, which we will use to create the rundict in the next step.

```
>>> idfs = pd.DataFrame({"file": files, "name": [file.basename() for file in files]})
```

The rundict, is the list of tasks we wish to do in parallel. This dictionary is passed to `parallel_process()`. Here, we want to execute `run_eplus()` with the following parameters:

```
>>> rundict = {
>>>     k: dict(
>>>         idfname=str(file),
>>>         prep_outputs=True,
>>>         weather_file=str(epw),
>>>         expandobjects=True,
>>>         verbose=True,
>>>         design_day=True,
>>>         simulate=True,
>>>     )
>>>     for k, file in idfs.file.to_dict().items()
>>> }
```

Finally, execute `parallel_process()`. The resulting `sql_file` paths, which we defined as the type of `output_report` attribute for `run_eplus()` is returned as a dictionary with the same keys as the index of the DataFrame.

```
>>> sql_files = parallel_process(rundict, run_eplus, use_kwargs=True, processors=-1)
>>> sql_files
{0: Path('cache/06e92da0247c71762d64aed4bcf3cdb2/output_data/
→06e92da0247c71762d64aed4bcf3cdb2out.sql'),
 1: Path('cache/ae8caf562b3519942ef88f533800dd0/output_data/
→ae8caf562b3519942ef88f533800dd0out.sql'),
 2: Path('cache/9d14a6aa6fda03a77ed5c5c48d28a73b/output_data/
→9d14a6aa6fda03a77ed5c5c48d28a73bout.sql'),
 3: Path('cache/5ddfa8827d2a577aabb02d60195bf53a/output_data/
→5ddfa8827d2a577aabb02d60195bf53aout.sql'),
 4: Path('cache/225c3428099e2abcc4051750db12731b/output_data/
→225c3428099e2abcc4051750db12731bout.sql'),
 5: Path('cache/0991d42c5af387833b68adffc0d7b523/output_data/
→0991d42c5af387833b68adffc0d7b523out.sql'),
 6: Path('cache/e10a4bf8bae93b0b0d2ad2638c807b61/output_data/
→e10a4bf8bae93b0b0d2ad2638c807b61out.sql'),
 7: Path('cache/86439047af9e8ff4650d6bab460d5e70/output_data/
→86439047af9e8ff4650d6bab460d5e70out.sql'),
 8: Path('cache/68da0886afa316f75bc63d7e576d0228/output_data/
→68da0886afa316f75bc63d7e576d0228out.sql'),
 9: Path('cache/68a8be47fe4573a61d388a0101798958/output_data/
→68a8be47fe4573a61d388a0101798958out.sql'),
10: Path('cache/f6f8abae5272bf607a9f53d18c10a50d/output_data/
→f6f8abae5272bf607a9f53d18c10a50dout.sql'),
11: Path('cache/4cf8589df098bb0c3f2b9f8589ec6ed6/output_data/
→4cf8589df098bb0c3f2b9f8589ec6ed6out.sql'),
12: Path('cache/5dd643faf859ed1aed5adffcecd0d47c/output_data/
→5dd643faf859ed1aed5adffcecd0d47cout.sql'),
13: Path('cache/e7cf6ae2be8917a409c9a1acad3bc349/output_data/
→e7cf6ae2be8917a409c9a1acad3bc349out.sql'),
14: Path('cache/3f122e04f7d8d19195cb8818a0be390f/output_data/
→3f122e04f7d8d19195cb8818a0be390fout.sql'),
15: Path('cache/d263b5b5d3bc56f2fb3795c61ac89cfe/output_data/
→d263b5b5d3bc56f2fb3795c61ac89cfeout.sql')}
```

1.7 Schedules

archetypal can parse EnergyPlus schedules. In EnergyPlus, there are many ways to define schedules in an IDF file. The Schedule module defines a class that handles parsing, plotting converting schedules.

1.7.1 Reading Schedules

archetypal can read almost any schedules defined in an IDF file using a few commands. First,

```
>>> import archetypal as ar
>>> idf = ar.IDF(<idf-file-path>)
>>> this_schedule = Schedule(Name='name', idf=idf)
```

1.7.2 Converting Schedules

Some tools typically rely on a group of 3 schedules; defined as a Yearly, Weekly, Daily schedule object. This is the case for the *IDF to UMI* converter and for the IDF to TRNSYS converter. The Schedule module of *archetypal* can handle this conversion.

The *year-week-day* representation for any schedule object is invoked with the `to_year_week_day()` method:

```
>>> this_schedule.to_year_week_day()
```

1.7.3 Plotting Schedules

Schedules can be parsed as `pandas.Series` objects (call the *series* property on a Schedule object) which then exposes useful methods from the pandas package. For convenience, a wrapper for the plotting method is built-in the Schedule class. To plot the full annual schedule (or a specific range), simply call the `archetypal.schedule.Schedule.plot()` method. For example,

```
>>> this_schedule.plot(slice=("2018/01/02", "2018/01/03"), drawstyle="steps-post")
```

1.8 Creating an Umi template

This tutorial explains how to create an Umi Template Library from scratch using *archetypal*. This section presents each required steps to create a valid Umi Template Library object. Every object will be presented with their default parameters, and simple examples will show how to create objects with the minimum required parameters. Before we start, here is a description of the overall structure of an Umi Template Library and the Building Templates it contains.

1.8.1 Umi Template Structure

An Umi Template is a collection of various other objects that are referenced between each other. At the top of the hierarchy, there is the `UmiTemplateLibrary` object which holds all the other bits and pieces making up the template library. The second level is therefore the `BuildingTemplate` object. There is one `BuildingTemplate` for each building models (or archetypes). Each `BuildingTemplate` is made of a series of children objects, and multiple `BuildingTemplates` can share the same children. For example, two buildings can share the same lighting schedule or the same opaque material.

For simplicity, this tutorial begins with the lowest level in the hierarchy (or the leaf in a graph structure): The materials.

1.8.2 Defining materials

The first step is to create the library of materials from which the constructions will be made. (used as *Layers* in constructions). There are `OpaqueMaterial`, `GlazingMaterial` and `GasMaterial` to define.

Opaque materials

Here are the parameters and their default values for an `OpaqueMaterial` object (see [OpaqueMaterial](#) for more information)

```
def __init__(
    Name,
    Conductivity,
    SpecificHeat,
    SolarAbsorptance=0.7,
    ThermalEmittance=0.9,
    VisibleAbsorptance=0.7,
    Roughness="Rough",
    Cost=0,
    Density=1,
    MoistureDiffusionResistance=50,
    EmbodiedCarbon=0.45,
    EmbodiedEnergy=0,
    TransportCarbon=0,
    TransportDistance=0,
    TransportEnergy=0,
    SubstitutionRatePattern=None,
    SubstitutionTimestep=20,
    **kwargs)
```

Users can keep the default values by simply omitting them in the constructor. For example, one can create a simple list of 4 `OpaqueMaterial` objects with default values. Note that the `Name`, `Conductivity` and `SpecificHeat` are required parameters:

```
concrete = ar.OpaqueMaterial(Name="Concrete", Conductivity=0.5, SpecificHeat=800,
    ↳Density=1500)
insulation = ar.OpaqueMaterial(Name="Insulation", Conductivity=0.04, SpecificHeat=1000,
    ↳Density=30)
brick = ar.OpaqueMaterial(Name="Brick", Conductivity=1, SpecificHeat=900, Density=1900)
plywood = ar.OpaqueMaterial(Name="Plywood", Conductivity=0.13, SpecificHeat=800,
    ↳Density=540)
```

Add these 4 materials to a variable named `OpaqueMaterials`. This variable will be referenced at the end when the `UmiTemplateLibrary` object will be created.

```
# List of OpaqueMaterial objects (needed for Umi template creation)
OpaqueMaterials = [concrete, insulation, brick, plywood]
```

Glazing materials

The same goes for the `GlazingMaterial` objects (see [GlazingMaterial](#) for more information)

```
def __init__(
    Name,
    Density=2500,
    Conductivity=0,
    SolarTransmittance=0,
    SolarReflectanceFront=0,
```

(continues on next page)

(continued from previous page)

```
SolarReflectanceBack=0,
VisibleTransmittance=0,
VisibleReflectanceFront=0,
VisibleReflectanceBack=0,
IRTransmittance=0,
IREmissivityFront=0,
IREmissivityBack=0,
DirtFactor=1.0,
Type=None,
Cost=0.0,
Life=1,
**kwargs)
```

A “Transparent Glass” object is created with the following optical and thermal properties:

```
glass = ar.GlazingMaterial(
    Name="Glass",
    Density=2500,
    Conductivity=1,
    SolarTransmittance=0.7,
    SolarReflectanceFront=0.5,
    SolarReflectanceBack=0.5,
    VisibleTransmittance=0.7,
    VisibleReflectanceFront=0.5,
    VisibleReflectanceBack=0.5,
    IRTransmittance=0.7,
    IREmissivityFront=0.5,
    IREmissivityBack=0.5)
```

The object is referenced in the following variable: .. code-block:: python

```
# List of GlazingMaterial objects (needed for Umi template creation) GlazingMaterials = [glass]
```

Gas materials

Here are all the parameters and their default values for a GasMaterial object (see [GasMaterial](#) for more information)

```
def __init__(
    Name,
    Cost=0,
    EmbodiedCarbon=0,
    EmbodiedEnergy=0,
    SubstitutionTimestep=100,
    TransportCarbon=0,
    TransportDistance=0,
    TransportEnergy=0,
    SubstitutionRatePattern=None,
    Conductivity=2.4,
    Density=2400,
    **kwargs)
```

Example of GasMaterial object:

```
air = ar.GasMaterial(Name="Air", Conductivity=0.02, Density=1.24)
# List of GasMaterial objects (needed for Umi template creation)
GasMaterials = [air]
```

1.8.3 Defining material layers

Once the materials are created, layers (or `MaterialLayer` objects) can be created. Here are the parameters and their default values for an `MaterialLayer` object

```
def __init__(Material, Thickness)
```

The `Material` (from *`OpaqueMaterial`* or *`GlazingMaterial`* or *`GasMaterial`*) and `Thickness` are required parameters:

```
concreteLayer = ar.MaterialLayer(concrete, Thickness=0.2)
insulationLayer = ar.MaterialLayer(insulation, Thickness=0.5)
brickLayer = ar.MaterialLayer(brick, Thickness=0.1)
plywoodLayer = ar.MaterialLayer(plywood, Thickness=0.016)
glassLayer = ar.MaterialLayer(glass, Thickness=0.16)
airLayer = ar.MaterialLayer(air, Thickness=0.04)
```

1.8.4 Defining constructions

Once the material layers are created, wall assemblies (or *`OpaqueConstruction`* objects) can be created.

Opaque constructions

Here are all the parameters and default values for an `OpaqueConstruction` object (see *`OpaqueConstruction`* for more information)

```
def __init__(
    Name,
    Layers,
    Surface_Type,
    Outside_Boundary_Condition,
    IsAdiabatic,
    **kwargs)
```

An `OpaqueConstruction` requires a few parameters such as the `Layers` (a list of *`OpaqueMaterial`* objects), the `Surface_Type` (choice of "Partition", "")

```
# OpaqueConstruction using OpaqueMaterial objects
wall_int = ar.OpaqueConstruction(
    Layers=[plywoodLayer],
    Surface_Type="Partition",
    Outside_Boundary_Condition="Zone",
    IsAdiabatic=True)

wall_ext = ar.OpaqueConstruction(
    Layers=[concreteLayer, insulationLayer, brickLayer],
```

(continues on next page)

(continued from previous page)

```
Surface_Type="Facade",
Outside_Boundary_Condition="Outdoors")

floor = ar.OpacityConstruction(
Layers=[concreteLayer, plywoodLayer],
Surface_Type="Ground",
Outside_Boundary_Condition="Zone")

roof = ar.OpacityConstruction(
Layers=[plywoodLayer, insulationLayer, brickLayer],
Surface_Type="Roof",
Outside_Boundary_Condition="Outdoors")
# List of OpacityConstruction objects (needed for Umi template creation)
OpacityConstructions = [wall_int, wall_ext, floor, roof]
```

Window constructions

Here are all the parameters and their default values for an WindowConstruction object (see [WindowConstruction](#) doc for more information)

```
def __init__(
    Layers,
    Category="Double",
    AssemblyCarbon=0,
    AssemblyCost=0,
    AssemblyEnergy=0,
    DisassemblyCarbon=0,
    DisassemblyEnergy=0,
    **kwargs)
```

Example of WindowConstruction object:

```
# WindowConstruction using GlazingMaterial and GasMaterial objects
window = ar.WindowConstruction(Layers=[glassLayer, airLayer, glassLayer])
# List of WindowConstruction objects (needed for Umi template creation)
WindowConstructions = [window]
```

Structure definition

Here are all the parameters and their default values for an StructureInformation object (see [StructureDefinition](#) doc for more information)

```
def __init__(
    *args,
    AssemblyCarbon=0,
    AssemblyCost=0,
    AssemblyEnergy=0,
    DisassemblyCarbon=0,
    DisassemblyEnergy=0,
    MassRatios=None,
    **kwargs)
```

We observe that `StructureInformation` uses `MassRatio` objects. Here are the parameters of `MassRatio` object (see [MassRatio](#) doc for more information)

```
def __init__(HighLoadRatio=None, Material=None, NormalRatio=None)
```

Example of `StructureInformation` object:

```
# StructureInformation using OpaqueMaterial objects
mass_ratio = ar.MassRatio(Material=plywood, HighLoadRatio=1, NormalRatio=1)
struct_definition = ar.StructureInformation(MassRatios=[mass_ratio])
# List of StructureInformation objects (needed for Umi template creation)
StructureDefinitions = [struct_definition]
```

1.8.5 Defining schedules

Creating Umi template objects to define schedules (e.g. `DaySchedule`).

- Day schedules

Here are all the parameters and their default values for a `DaySchedule` object (see [DaySchedule](#) doc for more information)

```
def __init__(
    Name=None,
    idf=None,
    start_day_of_the_week=0,
    strict=False,
    base_year=2018,
    schType=None,
    schTypeLimitsName=None,
    values=None,
    **kwargs)
```

Example of `DaySchedule` objects:

```
# Always on
sch_d_on = ar.DaySchedule.from_values(
    [1] * 24, Category="Day", schTypeLimitsName="Fractional", Name="AlwaysOn")
# Always off
sch_d_off = ar.DaySchedule.from_values(
    [0] * 24, Category="Day", schTypeLimitsName="Fractional", Name="AlwaysOff")
# DHW
sch_d_dhw = ar.DaySchedule.from_values(
    [0.3] * 24, Category="Day", schTypeLimitsName="Fractional", Name="DHW")
# Internal gains
sch_d_gains = ar.DaySchedule.from_values(
    [0] * 6 + [0.5, 0.6, 0.7, 0.8, 0.9, 1] + [0.7] * 6 + [0.4] * 6,
    Category="Day",
    schTypeLimitsName="Fractional",
    Name="Gains",)
# List of DaySchedule objects (needed for Umi template creation)
DaySchedules = [sch_d_on, sch_d_dhw, sch_d_gains, sch_d_off]
```

- Week schedules

Here are all the parameters and their default values for a *WeekSchedule* object (see *WeekSchedule* doc for more information)

```
def __init__(
    Name=None,
    idf=None,
    start_day_of_the_week=0,
    strict=False,
    base_year=2018,
    schType=None,
    schTypeLimitsName=None,
    values=None,
    **kwargs)
```

Example of *WeekSchedule* objects:

```
# WeekSchedules using DaySchedule objects
# Variable `days` needs a list of 7 dict,
# representing the 7 days of the week
sch_w_on = ar.WeekSchedule(
    days=[
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_on.id}],
    Category="Week",
    schTypeLimitsName="Fractional",
    Name="AlwaysOn")
# Always off
sch_w_off = ar.WeekSchedule(
    days=[
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_off.id}],
    Category="Week",
    schTypeLimitsName="Fractional",
    Name="AlwaysOff")
# DHW
sch_w_dhw = ar.WeekSchedule(
    days=[
        {"$ref": sch_d_dhw.id},
        {"$ref": sch_d_dhw.id},
        {"$ref": sch_d_dhw.id},
        {"$ref": sch_d_dhw.id},
        {"$ref": sch_d_dhw.id},
```

(continues on next page)

(continued from previous page)

```

        {"$ref": sch_d_dhw.id},
        {"$ref": sch_d_dhw.id},],
    Category="Week",
    schTypeLimitsName="Fractional",
    Name="DHW")
# Internal gains
sch_w_gains = ar.WeekSchedule(
    days=[
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},
        {"$ref": sch_d_gains.id},],
    Category="Week",
    schTypeLimitsName="Fractional",
    Name="Gains")
# List of WeekSchedule objects (needed for Umi template creation)
WeekSchedules = [sch_w_on, sch_w_off, sch_w_dhw, sch_w_gains]

```

WeekSchedule object can also be created from a dictionary. For example, we create a WeekSchedule *AlwaysOn* from a dictionary and using DaySchedule *AlwaysOn* objects:

```

# Dict of a WeekSchedule (like it would be written in json file)
dict_w_on = {
    "Category": "Week",
    "Days": [
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_on.id},
        {"$ref": sch_d_off.id},
        {"$ref": sch_d_on.id},
    ],
    "Type": "Fraction",
    "Name": "OnOff_2"}
# Creates WeekSchedule from dict (from json)
sch_w_on = ar.WeekSchedule.from_dict(**dict_w_on)

```

- Year schedules

Here are all the parameters and their default values for an YearSchedule object (see [YearSchedule](#) doc for more information)

```

def __init__(
    Name=None,
    idf=None,
    start_day_of_the_week=0,
    strict=False,
    base_year=2018,
    schType=None,

```

(continues on next page)

(continued from previous page)

```
schTypeLimitsName=None,
values=None,
**kwargs)
```

YearSchedule are created from dictionaries. For example, we create YearSchedules from dictionaries and using WeekSchedule objects:

```
# YearSchedules using DaySchedule objects
# Always on
dict_on = {
    "Category": "Year",
    "Parts": [
        {
            "FromDay": 1,
            "FromMonth": 1,
            "ToDay": 31,
            "ToMonth": 12,
            "Schedule": {"$ref": sch_w_on.id}
        }
    ],
    "Type": "Fraction",
    "Name": "AlwaysOn"}
sch_y_on = ar.YearSchedule.from_dict(**dict_on)
# Always off
dict_off = {
    "Category": "Year",
    "Parts": [
        {
            "FromDay": 1,
            "FromMonth": 1,
            "ToDay": 31,
            "ToMonth": 12,
            "Schedule": {"$ref": sch_w_off.id}}],
    "Type": "Fraction",
    "Name": "AlwaysOff"}
sch_y_off = ar.YearSchedule.from_dict(**dict_off)
# Year ON/OFF
dict_on_off = {
    "Category": "Year",
    "Parts": [
        {
            "FromDay": 1,
            "FromMonth": 1,
            "ToDay": 31,
            "ToMonth": 5,
            "Schedule": {"$ref": sch_w_on.id}
        },
        {
            "FromDay": 1,
            "FromMonth": 6,
            "ToDay": 31,
            "ToMonth": 12,
            "Schedule": {"$ref": sch_w_off.id}
        }
    ]
}
```

(continues on next page)

(continued from previous page)

```

    }
    ],
    "Type": "Fraction",
    "Name": "ON_OFF"}
sch_y_on_off = ar.YearSchedule.from_dict(**dict_on_off)
# DHW
dict_dhw = {
    "Category": "Year",
    "Parts": [
        {
            "FromDay": 1,
            "FromMonth": 1,
            "ToDay": 31,
            "ToMonth": 12,
            "Schedule": {"$ref": sch_w_dhw.id}},
    "Type": "Fraction",
    "Name": "DHW"}
sch_y_dhw = ar.YearSchedule.from_dict(**dict_dhw)
# Internal gains
dict_gains = {
    "Category": "Year",
    "Parts": [
        {
            "FromDay": 1,
            "FromMonth": 1,
            "ToDay": 31,
            "ToMonth": 12,
            "Schedule": {"$ref": sch_w_gains.id}},
    "Type": "Fraction",
    "Name": "Gains"}
sch_y_gains = ar.YearSchedule.from_dict(**dict_gains)
# List of YearSchedule objects (needed for Umi template creation)
YearSchedules = [sch_y_on, sch_y_off, sch_y_on_off, sch_y_dhw, sch_y_gains]

```

1.8.6 Defining window settings

Creating Umi template objects to define window settings

Here are all the parameters and their default values for an WindowSetting object (see [WindowSetting](#) doc for more information)

```

def __init__(
    Name,
    Construction=None,
    OperableArea=0.8,
    AfnWindowAvailability=None,
    AfnDischargeC=0.65,
    AfnTempSetpoint=20,
    IsVirtualPartition=False,
    IsShadingSystemOn=False,
    ShadingSystemAvailabilitySchedule=None,

```

(continues on next page)

(continued from previous page)

```

ShadingSystemSetpoint=180,
ShadingSystemTransmittance=0.5,
ShadingSystemType=0,
Type=WindowType.External,
IsZoneMixingOn=False,
ZoneMixingAvailabilitySchedule=None,
ZoneMixingDeltaTemperature=2,
ZoneMixingFlowRate=0.001,
**kwargs)

```

Example of WindowSetting object:

```

# WindowSetting using WindowConstruction and YearSchedule objects
window_setting = ar.WindowSetting(
    Name="window_setting_1",
    Construction=window,
    AfnWindowAvailability=sch_y_off,
    ShadingSystemAvailabilitySchedule=sch_y_off,
    ZoneMixingAvailabilitySchedule=sch_y_off)
# List of WindowSetting objects (needed for Umi template creation)
WindowSettings = [window_setting]

```

1.8.7 Defining DHW settings

Creating Umi template objects to define DHW settings

Here are all the parameters and their default values for an DomesticHotWaterSetting object (see [DomesticHotWaterSetting](#) doc for more information)

```

def __init__(
    Name,
    IsOn=True,
    WaterSchedule=None,
    FlowRatePerFloorArea=0.03,
    WaterSupplyTemperature=65,
    WaterTemperatureInlet=10,
    **kwargs)

```

Example of DomesticHotWaterSetting object:

```

# DomesticHotWaterSetting using YearSchedule objects
dhw_setting = ar.DomesticHotWaterSetting(
    Name="dhw_setting_1",
    IsOn=True,
    WaterSchedule=sch_y_dhw,
    FlowRatePerFloorArea=0.03,
    WaterSupplyTemperature=65,
    WaterTemperatureInlet=10,)
# List of DomesticHotWaterSetting objects (needed for Umi template creation)
DomesticHotWaterSettings = [dhw_setting]

```

1.8.8 Defining ventilation settings

Creating Umi template objects to define ventilation settings

Here are all the parameters and their default values for an `VentilationSetting` object (see [VentilationSetting](#) doc for more information)

```
def __init__(
    Name,
    NatVentSchedule=None,
    ScheduledVentilationSchedule=None,
    Afn=False,
    Infiltration=0.1,
    IsBuoyancyOn=True,
    IsInfiltrationOn=True,
    IsNatVentOn=False,
    IsScheduledVentilationOn=False,
    IsWindOn=False,
    NatVentMaxOutdoorAirTemp=30,
    NatVentMaxRelHumidity=90,
    NatVentMinOutdoorAirTemp=0,
    NatVentZoneTempSetpoint=18,
    ScheduledVentilationAch=0.6,
    ScheduledVentilationSetpoint=18,
    **kwargs)
```

Example of `VentilationSetting` object:

```
# VentilationSetting using YearSchedule objects
vent_setting = ar.VentilationSetting(
    Name="vent_setting_1",
    NatVentSchedule=sch_y_off,
    ScheduledVentilationSchedule=sch_y_off,)
# List of VentilationSetting objects (needed for Umi template creation)
VentilationSettings = [vent_setting]
```

1.8.9 Defining zone conditioning settings

Creating Umi template objects to define zone conditioning settings

Here are all the parameters and their default values for an `ZoneConditioning` object (see [ZoneConditioning](#) doc for more information)

```
def __init__(
    Name,
    CoolingCoeffOfPerf=1,
    CoolingLimitType="NoLimit",
    CoolingSetpoint=26,
    CoolingSchedule=None,
    EconomizerType="NoEconomizer",
    HeatRecoveryEfficiencyLatent=0.65,
    HeatRecoveryEfficiencySensible=0.7,
    HeatRecoveryType="None",
```

(continues on next page)

(continued from previous page)

```

HeatingCoeffOfPerf=1,
HeatingLimitType="NoLimit",
HeatingSetpoint=20,
HeatingSchedule=None,
IsCoolingOn=True,
IsHeatingOn=True,
IsMechVentOn=True,
MaxCoolFlow=100,
MaxCoolingCapacity=100,
MaxHeatFlow=100,
MaxHeatingCapacity=100,
MinFreshAirPerArea=0,
MinFreshAirPerPerson=0.00944,
MechVentSchedule=None,
**kwargs)

```

Example of ZoneConditioning object:

```

# ZoneConditioning using YearSchedule objects
zone_conditioning = ar.ZoneConditioning(
    Name="conditioning_setting_1",
    CoolingSchedule=sch_y_on,
    HeatingSchedule=sch_y_on,
    MechVentSchedule=sch_y_off,)
# List of ZoneConditioning objects (needed for Umi template creation)
ZoneConditionings = [zone_conditioning]

```

1.8.10 Defining zone construction sets

Creating Umi template objects to define zone construction sets

Here are all the parameters and their default values for an ZoneConstructionSet object (see [ZoneConstructionSet](#) doc for more information)

```

def __init__(
    *args,
    Zone_Names=None,
    Slab=None,
    IsSlabAdiabatic=False,
    Roof=None,
    IsRoofAdiabatic=False,
    Partition=None,
    IsPartitionAdiabatic=False,
    Ground=None,
    IsGroundAdiabatic=False,
    Facade=None,
    IsFacadeAdiabatic=False,
    **kwargs)

```

Example of ZoneConstructionSet objects:

```
# ZoneConstructionSet using OpaqueConstruction objects
# Perimeter zone
zone_constr_set_perim = ar.ZoneConstructionSet(
    Name="constr_set_perim",
    Slab=floor,
    Roof=roof,
    Partition=wall_int,
    Ground=floor,
    Facade=wall_ext)
# Core zone
zone_constr_set_core = ar.ZoneConstructionSet(
    Name="constr_set_core",
    Slab=floor,
    Roof=roof,
    Partition=wall_int,
    IsPartitionAdiabatic=True,
    Ground=floor,
    Facade=wall_ext)
# List of ZoneConstructionSet objects (needed for Umi template creation)
ZoneConstructionSets = [zone_constr_set_perim, zone_constr_set_core]
```

1.8.11 Defining zone loads

Creating Umi template objects to define zone loads

Here are all the parameters and their default values for an `ZoneLoad` object (see `ZoneLoad` doc for more information)

```
def __init__(
    Name,
    DimmingType="Continuous",
    EquipmentAvailabilitySchedule=None,
    EquipmentPowerDensity=12,
    IlluminanceTarget=500,
    LightingPowerDensity=12,
    LightsAvailabilitySchedule=None,
    OccupancySchedule=None,
    IsEquipmentOn=True,
    IsLightingOn=True,
    IsPeopleOn=True,
    PeopleDensity=0.2,
    **kwargs)
```

Example of `ZoneLoad` object:

```
# ZoneLoad using YearSchedule objects
zone_load = ar.ZoneLoad(
    Name="zone_load_1",
    EquipmentAvailabilitySchedule=sch_y_gains,
    LightsAvailabilitySchedule=sch_y_gains,
    OccupancySchedule=sch_y_gains)
# List of ZoneLoad objects (needed for Umi template creation)
```

(continues on next page)

(continued from previous page)

```
ZoneLoads = [zone_load]
```

1.8.12 Defining zones

Creating Umi template objects to define zones

Here are all the parameters and their default values for an `Zone` object (see [Zone](#) doc for more information)

```
def __init__(
    Name,
    Conditioning=None,
    Constructions=None,
    DomesticHotWater=None,
    Loads=None,
    Ventilation=None,
    Windows=None,
    InternalMassConstruction=None,
    InternalMassExposedPerFloorArea=1.05,
    DaylightMeshResolution=1,
    DaylightWorkplaneHeight=0.8,
    **kwargs)
```

Example of `Zone` objects:

```
# Zones using ZoneConditioning, ZoneConstructionSet, DomesticWaterSetting,
# ZoneLoad, VentilationSetting, WindowSetting and OpaqueConstruction objects
# Perimeter zone
perim = ar.Zone(
    Name="Perim_zone",
    Conditioning=zone_conditioning,
    Constructions=zone_constr_set_perim,
    DomesticHotWater=dhw_setting,
    Loads=zone_load,
    Ventilation=vent_setting,
    Windows=window_setting,
    InternalMassConstruction=wall_int)
# Core zone
core = ar.Zone(
    Name="Core_zone",
    Conditioning=zone_conditioning,
    Constructions=zone_constr_set_core,
    DomesticHotWater=dhw_setting,
    Loads=zone_load,
    Ventilation=vent_setting,
    Windows=window_setting,
    InternalMassConstruction=wall_int)
# List of Zone objects (needed for Umi template creation)
Zones = [perim, core]
```

1.8.13 Defining building template

Creating Umi template objects to define building template

Here are all the parameters and their default values for an BuildingTemplate object (see [BuildingTemplate](#) doc for more information)

```
def __init__(
    Name,
    Core=None,
    Perimeter=None,
    Structure=None,
    Windows=None,
    Lifespan=60,
    PartitionRatio=0.35,
    DefaultWindowToWallRatio=0.4,
    **kwargs)
```

Example of BuildingTemplate object:

```
# BuildingTemplate using Zone, StructureInformation and WindowSetting objects
building_template = ar.BuildingTemplate(
    Name="Building_template_1",
    Core=core,
    Perimeter=perim,
    Structure=struct_definition,
    Windows=window_setting,)
# List of BuildingTemplate objects (needed for Umi template creation)
BuildingTemplates = [building_template]
```

1.8.14 Creating Umi template

Creating Umi template from all objects defined before (see [UmiTemplate](#) doc for more information)

Example of BuildingTemplate object:

```
# UmiTemplateLibrary using all lists of objects created before
umi_template = ar.UmiTemplateLibrary(
    name="unnamed",
    BuildingTemplates=BuildingTemplates,
    GasMaterials=GasMaterials,
    GlazingMaterials=GlazingMaterials,
    OpaqueConstructions=OpaqueConstructions,
    OpaqueMaterials=OpaqueMaterials,
    WindowConstructions=WindowConstructions,
    StructureDefinitions=StructureDefinitions,
    DaySchedules=DaySchedules,
    WeekSchedules=WeekSchedules,
    YearSchedules=YearSchedules,
    DomesticHotWaterSettings=DomesticHotWaterSettings,
    VentilationSettings=VentilationSettings,
    WindowSettings=WindowSettings,
    ZoneConditionings=ZoneConditionings,
```

(continues on next page)

(continued from previous page)

```
ZoneConstructionSets=ZoneConstructionSets,  
ZoneLoads=ZoneLoads,  
Zones=Zones,  
)
```

And finally we use this following line of code to create the json file that can be imported into Umi as a template:

```
umi_template.to_dict()
```

1.9 Editing UMI Template Files

archetypal can read an UMI Template File using the command:

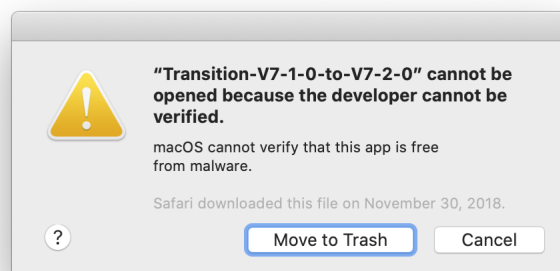
```
>>> from archetypal import UmiTemplateLibrary  
>>> template_library = UmiTemplateLibrary.open()
```

which returns an *UmiTemplateLibrary* object.

1.10 Troubleshooting

1.10.1 MacOS Catalina Compatibility

With the release of MacOS Catalina, Apple requires apps to be signed and/or notarized. This helps users make sure they open apps from trusted developers. It appears that the transition files that are needed to upgrade older IDF files to the latest version of EnergyPlus are not signed. Thus, users using *archetypal* on the MacOS platform might encounter an error of this type: “Transition cannot be opened because the developer cannot be verified”. (see [Missing transition programs](#) for more details on downloading and installing older transition programs).



It seems that clicking “cancel” will still work, although the prompt will appear for all older transition files repetitively. An issue has been submitted [here](#) on the EnergyPlus github repository. Hopefully, the developers of EnergyPlus will be able to address this issue soon.

1.10.2 Missing transition programs

For older EnergyPlus file versions (< 7-1-0), the necessary transition files are not included with the EnergyPlus installer. Users must download and install missing transition programs manually. This can be achieved in two simple steps:

1. Navigate to the EnergyPlus [Knowledgebase](#) and download the appropriate transition programs depending on the platform you are using (MacOs, Linux or Windows). These programs come in the form of a zipped folder.
2. Extract all the files from the zipped folder to your EnergyPlus installation folder at `./PreProcess/IDFVersionUpdater/`. For example, on MacOs with EnergyPlus version 8-9-0, this path is `/Applications/EnergyPlus-8-9-0/PreProcess/IDFVersionUpdater/`.

1.11 Command reference

Archetypal provides many commands for managing packages and environments. The links on this page provide help for each command. You can also access help from the command line with the `--help` flag:

```
archetypal --help
```

1.12 Modules

1.12.1 IDF Class

<i>IDF</i>	Class for loading and parsing idf models.
<i>Outputs</i>	Handles preparation of EnergyPlus outputs.
<i>Meters</i>	Lists available meters in the IDF model.
<i>Variables</i>	Lists available variables in the IDF model.

archetypal.idfclass.IDF

```
class archetypal.idfclass.IDF(idfname: Optional[Union[str, IO, path.Path]] = None, epw=None,
                             as_version: Union[str,
                             archetypal.eplus_interface.version.EnergyPlusVersion] = '9-2-0',
                             annual=False, design_day=False, expandobjects=False, convert=False,
                             verbose=False, readvars=True, prep_outputs=True, include=None,
                             custom_processes=None, output_suffix='L', epmacro=False,
                             keep_data=True, keep_data_err=True, position=0, name=None,
                             output_directory=None, outputtype='standard', iddname:
                             Optional[Union[str, IO, path.Path]] = None, **kwargs)
```

Class for loading and parsing idf models.

This is the starting point to run simulations and retrieving results.

Wrapper over the `geomeppy.IDF` class and subsequently the `eppy.modeleditor.IDF` class.

Initialize an IDF object.

Parameters

- `output_directory` –

- **idfname** (*str* or *os.PathLike*) – The path of the idf file to read. If none, an in-memory IDF object is generated.
- **epw** (*str* or *os.PathLike*) – The weather-file. If None, epw can be specified in `IDF.simulate()`.
- **as_version** (*str*) – Specify the target EnergyPlus version for the IDF model. If `as_version` is higher than the file version, the model will be transitioned to the target version. This will not overwrite the file unless `IDF.save()` is invoked. See `IDF.save()`, `IDF.saveas()` and `IDF.savecopy()` for other IO operations on IDF objects.
- **annual** (*bool*) – If True then force annual simulation (default: False).
- **design_day** (*bool*) – Force design-day-only simulation (default: False).
- **expandobjects** (*bool*) – Run ExpandObjects prior to simulation (default: True).
- **convert** (*bool*) – If True, only convert IDF->epJSON or epJSON->IDF.
- **outputtype** (*str*) – Specifies the idf string representation of the model. Choices are: “standard”, “nocomment1”, “nocomment2”, “compressed”.

EnergyPlus args: `tmp_dir=None`, `as_version=None`, `prep_outputs=True`, `include=None`, `keep_original=True`,

classmethod `from_example_files(example_name, epw=None, **kwargs)`

Load an IDF model from the ExampleFiles folder by name.

Examples

```
idf = IDF.from_example_files( "minimal", "USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw"
)
```

Parameters

- **example_name** (*str*) – The name of the example file to load.
- **epw** (*str*) – The name of the weather file contained the WeatherData folder or the path to a specific weather file.
- ****kwargs** – keyword arguments passed to the IDF constructor.

Returns An IDF model.

Return type (*IDF*)

setiddname(*iddname*, *testing=False*)

Set EnergyPlus IDD path for model.

Sets the version of EnergyPlus which is to be used by eppy.

Parameters

- **iddname** (*str*) – Path to the IDD file.
- **testing** (*bool*) –

read()

Read the IDF file and the IDD file.

If the IDD file had already been read, it will not be read again.

Read populates the following data structures:

- `idfobjects` : list
- `model` : list
- `idd_info` : list
- `idd_index` : dict

getiddname()

Get the name of the current IDD used by eppy.

setidd(*iddinfo*, *iddindex*, *block*, *idd_version*)

Set the IDD to be used by eppy.

Parameters

- **iddindex** –
- **iddinfo** (*list*) – Comments and metadata about fields in the IDD.
- **block** (*list*) – Field names in the IDD.

property block: list

Set EnergyPlus field ID names of the IDF from the IDD.

property idd_info: list

Set comments and metadata about fields in the IDD.

property idd_index: dict

Set pair of dicts used for fast lookups of names of groups of objects.

property idfobjects: dict

Set dict of lists of `idf_MSequence` objects in the IDF.

property model: eppy.EPlusInterfaceFunctions.eplusdata.Eplusdata

Set Eplusdata object containing representations of IDF objects.

property idd_version: tuple

Return the version of the iddname as a tuple.

property iddname: path.Path

Get or set the iddname path used to parse the idf model.

property file_version: archetypal.eplus_interface.version.EnergyPlusVersion

Return the EnergyPlusVersion of the idf text file.

property custom_processes: list

Return list of callables. Called on the output files.

property include: list

Return list of external files attached to model.

property keep_data_err: bool

If True, error files are copied back into `self.output_folder`.

Type `bool`

property keep_data: bool

If True, keep data folder.

Type `bool`

property output_suffix: str

L).

- L: Legacy (e.g., `eplustbl.csv`)

- C: Capital (e.g., eplusTable.csv)
- D: Dash (e.g., eplus-table.csv)

Type `str`

Type The suffix style for output file names (default

property idfname: `Union[path.Path, _io.StringIO]`

The path of the active (parsed) idf model.

Type `Path`

property epw: `path.Path`

The weather file path.

Type `Path`

property verbose

If True, print outputs to logging module.

See also:

[*archetypal.utils.config*](#)

Type `bool`

property expandobjects: `bool`

If True, run ExpandObjects prior to simulation.

Type `bool`

property readvars: `bool`

If True, run ReadVarsESO after simulation.

Type `bool`

property epmacro: `bool`

If True, run EPMacro prior to simulation.

Type `bool`

property design_day: `bool`

If True, force design-day-only simulation.

Type `bool`

property annual: `bool`

If True, force annual simulation.

Type `bool`

property convert: `bool`

If True, only convert IDF->epJSON or epJSON->IDF.

Dependent on input file type. No simulation.

Type `bool`

property prep_outputs

Bool or set list of custom outputs.

property as_version

Specify the desired EnergyPlusVersion for the IDF model.

property output_directory: `path.Path`

The output directory based on the hashing of the original file.

Notes

The hashing is performed before transitions or modifications. The directory is not created! Use `self.output_directory.mkdir_p()` to create it without an error if it exists.

Type `Path`

property output_prefix: `str`

eplus).

Type `str`

Type Prefix for output file names (default

property sim_id: `str`

The unique Id of the simulation.

Based on a subset of hashed variables:

- The idf model itself.
- epw
- annual
- design_day
- readvars
- as_version

Type `str`

property sim_info: `Optional[pandas.core.frame.DataFrame]`

Unique number generated for a simulation.

Type `DataFrame`

property sim_timestamp: `Union[str, pandas.core.series.Series]`

Return the simulation timestamp or “Never” if not ran yet.

property position: `int`

Position for the progress bar.

Type `int`

property idfversionupdater_dir: `path.Path`

The path of the IDFVersionUpdater folder.

Uses the current module’s `ep_version`.

Type `Path`

property name: `str`

Name of the idf model.

Can include the extension (.idf).

Type `str`

sql() → `dict`

Get the sql table report.

htm() → dict

Get the htm table report.

property energyplus_its: int

Return number of iterations needed to complete simulation.

open_htm()

Open .htm file in browser.

open_idf()

Open file in correct version of Ep-Launch.

open_last_simulation()

Open last simulation in Ep-Launch.

open_err()

Open last simulation err file in texteditor.

open_mdd()

Open .mdd file in browser.

This file shows all the report meters along with their “availability” for the current input file.

open_mtd()

Open .mtd file in browser.

This file contains the “meter details” for the run. This shows what report variables are on which meters and vice versa – which meters contain what report variables.

property sql_file

Get the sql file path.

property mtd_file: path.Path

Get the mtd file path.

property net_conditioned_building_area: float

Return the total conditioned area of a building.

Takes into account zone multipliers.

property unconditioned_building_area: float

Return the Unconditioned Building Area.

property total_building_area: float

Return the Total Building Area.

property partition_ratio: float

Lineal meters of partitions per m2 of floor area.

Type float

add_block(*args: Any, **kwargs: Any) → None

Add a block to the IDF.

Parameters

- **name** – A name for the block.
- **coordinates** – A list of (x, y) tuples representing the building outline.
- **height** – The height of the block roof above ground level.
- **num_stories** – The total number of stories including basement stories. Default : 1.
- **below_ground_stories** – The number of stories below ground. Default : 0.

- **below_ground_storey_height** – The height of each basement storey. Default : 2.5.
- **zoning** – The zoning pattern of the block. Default : by_storey
- **perim_depth** – Depth of the perimeter zones if the core/perim zoning pattern is requested. Default : 3.0.

add_shading_block(*args: Any, **kwargs: Any) → None

Add a shading block to the IDF.

Parameters

- **name** – A name for the block.
- **coordinates** – A list of (x, y) tuples representing the building outline.
- **height** – The height of the block roof above ground level.
- **num_stories** – The total number of stories including basement stories. Default : 1.
- **below_ground_stories** – The number of stories below ground. Default : 0.
- **below_ground_storey_height** – The height of each basement storey. Default : 2.5.

add_zone(zone: *geomeppy.builder.Zone*) → None

Add a zone to the IDF.

Parameters **zone** – A Zone object holding details about the zone.

bounding_box() → *geomeppy.geom.polygons.Polygon2D*

Calculate the site bounding box.

Returns A polygon of the bounding box.

property centroid

Calculate the centroid of the site bounding box.

Returns The centroid of the site bounding box.

copyidfobject(idfobject: *geomeppy.patches.EpBunch*) → *geomeppy.patches.EpBunch*

Add an IDF object to the IDF.

This has been monkey-patched to add the return value.

Parameters **idfobject** – The IDF object to copy. Usually from another IDF, or it can be used to copy within this IDF.

Returns EpBunch object.

getextensibleindex(key, name)

Get the index of the first extensible item.

Only for internal use. # TODO : hide this

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- **name** (*str*) – The name of the object to fetch.

Returns

Return type *int*

getiddgroupdict()

Return a idd group dictionary sample: {‘Plant-Condenser Loops’: [‘PlantLoop’, ‘Condenser-Loop’], ‘Compliance Objects’: [‘Compliance:Building’], ‘Controllers’: [‘Controller:WaterCoil’, ‘Controller:OutdoorAir’, ‘Controller:MechanicalVentilation’, ‘AirLoopHVAC:ControllerList’], ... }

Returns**Return type** `dict`**getobject**(*key*, *name*)

Fetch an IDF object given key and name.

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- **name** (*str*) – The name of the object to fetch.

Returns**Return type** EpBunch object.**getshadingsurfaces**(*surface_type*: *str* = "") → Union[List[`eppy.bunch_subclass.EpBunch`],
`eppy.idf_msequence.Idf_MSequence`]

Return all subsurfaces in the IDF.

Parameters **surface_type** – Type of surface to get. Defaults to all.**Returns** IDF surfaces.**getsubsurfaces**(*surface_type*: *str* = "") → Union[List[`eppy.bunch_subclass.EpBunch`],
`eppy.idf_msequence.Idf_MSequence`]

Return all subsurfaces in the IDF.

Parameters **surface_type** – Type of surface to get. Defaults to all.**Returns** IDF surfaces.**getsurfaces**(*surface_type*: *str* = "") → Union[List[`eppy.bunch_subclass.EpBunch`],
`eppy.idf_msequence.Idf_MSequence`]

Return all surfaces in the IDF.

Parameters **surface_type** – Type of surface to get. Defaults to all.**Returns** IDF surfaces.**idfstr**()

String representation of the IDF.

Returns**Return type** `str`**initnew**(*fname*)

Use the current IDD and create a new empty IDF. If the IDD has not yet been initialised then this is done first.

Parameters **fname** (*str*, *optional*) – Path to an IDF. This does not need to be set at this point.**initread**(*idfname*)

Use the current IDD and read an IDF from file. If the IDD has not yet been initialised then this is done first.

Parameters **idf_name** (*str*) – Path to an IDF file.**initreadtxt**(*idftxt*)

Use the current IDD and read an IDF from text data. If the IDD has not yet been initialised then this is done first.

Parameters **idftxt** (*str*) – Text representing an IDF file.**intersect**() → `None`

Intersect all surfaces in the IDF.

intersect_match() → *None*

Intersect all surfaces in the IDF, then set boundary conditions.

match() → *None*

Set boundary conditions for all surfaces in the IDF.

new(*fname=None*)

Create a blank new idf file. Filename is optional.

Parameters **fname** (*str*, *optional*) – Path to an IDF. This does not need to be set at this point.

popidfobject(*key, index*)

Pop an IDF object from the IDF.

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- **index** (*int*) – The index of the object to pop.

Returns

Return type EpBunch object.

printidf()

Print the IDF.

removeextensibles(*key, name*)

Remove extensible items in the object of key and name.

Only for internal use. # TODO : hide this

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- **name** (*str*) – The name of the object to fetch.

Returns

Return type EpBunch object

rotate(*angle: float, anchor: Optional[Union[geom.pygeom.vectors.Vector2D, geom.pygeom.vectors.Vector3D]] = None*) → *None*

Rotate the IDF counterclockwise by the angle given.

Parameters

- **angle** – Angle (in degrees) to rotate by.
- **anchor** – Point around which to rotate. Default is the centre of the the IDF's bounding box.

run(***kwargs*)

This method wraps the following method:

rruunn(*idf=None, weather=None, output_directory="", annual=False, design_day=False, idd=None, epmacro=False, e*

Wrapper around the EnergyPlus command line interface.

idf [*str*] Full or relative path to the IDF file to be run, or an IDF object.

weather [*str*] Full or relative path to the weather file.

output_directory [*str, optional*] Full or relative path to an output directory (default: 'run_outputs')

annual [*bool, optional*] If True then force annual simulation (default: False)

design_day [*bool, optional*] Force design-day-only simulation (default: False)

idd [str, optional] Input data dictionary (default: Energy+.idd in EnergyPlus directory)

epmacro [str, optional] Run EPMacro prior to simulation (default: False).

expandobjects [bool, optional] Run ExpandObjects prior to simulation (default: False)

readvars [bool, optional] Run ReadVarsESO after simulation (default: False)

output_prefix [str, optional] Prefix for output file names (default: eplus)

output_suffix [str, optional]

Suffix style for output file names (default: L) L: Legacy (e.g., eplustbl.csv) C: Capital (e.g., eplusTable.csv) D: Dash (e.g., eplus-table.csv)

version [bool, optional] Display version information (default: False)

verbose: str

Set verbosity of runtime messages (default: v) v: verbose q: quiet

ep_version: str EnergyPlus version, used to find install directory. Required if run() is called with an IDF file path rather than an IDF object.

str : status

CalledProcessError

AttributeError If no ep_version parameter is passed when calling with an IDF file path rather than an IDF object.

scale(factor: float, anchor: Optional[Union[geom.pygeom.vectors.Vector2D, geom.pygeom.vectors.Vector3D]] = None, axes: str = 'xy') → None

Scale the IDF by a scaling factor.

Parameters

- **factor** – Factor to scale by.
- **anchor** – Point to scale around. Default is the centre of the the IDF's bounding box.
- **axes** – Axes to scale on. Default 'xy'.

property simulation_files: list

The list of files generated by the simulation.

Type list

to_obj(fname: Optional[str] = None, mtl: Optional[str] = None) → None

Export an OBJ file representation of the IDF.

This can be used for viewing in tools which support the .obj format.

Parameters

- **fname** – A filename for the .obj file. If None we try to base it on IDF.idfname and change the filetype.
- **mtl** – The name of a .mtl file to be referenced from the .obj file. If None, we use default.mtl.

translate(vector: geom.pygeom.vectors.Vector2D) → None

Move the IDF in the direction given by a vector.

Parameters **vector** – A vector to translate by.

translate_to_origin() → None

Move an IDF close to the origin so that it can be viewed in SketchUp.

view_model(*test: Optional[bool] = False*) → *None*

Show a zoomable, rotatable representation of the IDF.

property simulation_dir: *path.Path*

The path where simulation results are stored.

Type *Path*

property schedules_dict: *dict*

schedule in the model.

Type Return the dict of {NAME

property outputs: *archetypal.idfclass.outputs.Outputs*

Return the Outputs class associated with the model.

property day_of_week_for_start_day

Get day of week for start day for the first found RUNPERIOD.

Monday = 0 .. Sunday = 6

property meters: *archetypal.idfclass.meters.Meters*

List of available meters for the *IDF* model.

The *IDF* model must be simulated once (to retrieve the .mdd file).

The listed meters may or may not be included in the idf file. If they are not, the output is added to the file and the model is simulated again. The output is appended to the *IDF.idfobjects* list, but will not overwrite the original idf file, unless *IDF.save()* is called.

Hint: Call *idf.meters.<output_group>.<meter_name>.values()* to retrieve a time-series based on the *pandas.Series* class which can be plotted.

See *Meter* and *EnergySeries* for more information.

Example

The *IDF.meters* attribute is populated with meters categories (*Output:Meter* or *Output:Meter:Cumulative*) and each category is populated with all the available meters.

```
>>> IDF.meters.OutputMeter.WaterSystems__MainsWater
>>> IDF.meters.OutputMeterCumulative.WaterSystems__MainsWater
```

property variables

List of available meters for the *IDF* model.

The *IDF* model must be simulated once (to retrieve the .mdd file).

The listed meters may or may not be included in the idf file. If they are not, the output is added to the file and the model is simulated again. The output is appended to the *IDF.idfobjects* list, but will not overwrite the original idf file, unless *IDF.save()* is called.

Hint: Call *idf.meters.<output_group>.<meter_name>.values()* to retrieve a time-series based on the *pandas.Series* class which can be plotted.

See *Meter* and *EnergySeries* for more information.

Example

The `IDF.meters` attribute is populated with meters categories (*Output:Meter* or *Output:Meter:Cumulative*) and each category is populated with all the available meters.

```
>>> IDF.variables.OutputVariable
>>> IDF.variables.OutputVariable
```

simulate(*force=False, **kwargs*)

Execute EnergyPlus.

Specified kwargs overwrite IDF parameters. ExpandObjects, Basement and Slab preprocessors are ran before EnergyPlus.

Does not return anything.

Parameters **force** (*bool*) – Force simulation even though results exist in *self.simulation_dir*.

Keyword Arguments

- **eplus_file** (*str*) – path to the idf file.
- **weather_file** (*str*) – path to the EPW weather file.
- **output_directory** (*str, optional*) – path to the output folder.
- **ep_version** (*str, optional*) – EnergyPlus executable version to use, eg: 9-2-0
- **output_report** – ‘sql’ or ‘htm’
- **prep_outputs** (*bool or list, optional*) – if True, meters and variable outputs will be appended to the idf files. Can also specify custom outputs as list of ep-object outputs.
- **keep_data** (*bool*) – If True, files created by EnergyPlus are saved to the tmp_dir.
- **annual** (*bool*) – If True then force annual simulation (default: False)
- **design_day** (*bool*) – Force design-day-only simulation (default: False)
- **epmacro** (*bool*) – Run EPMacro prior to simulation (default: False)
- **expandobjects** (*bool*) – Run ExpandObjects prior to simulation (default: True)
- **readvars** (*bool*) – Run ReadVarsESO after simulation (default: False)
- **output_prefix** (*str, optional*) – Prefix for output file names.
- **output_suffix** (*str, optional*) – Suffix style for output file names (default: L)
Choices are:
 - L: Legacy (e.g., eplustbl.csv)
 - C: Capital (e.g., eplusTable.csv)
 - D: Dash (e.g., eplus-table.csv)
- **version** (*bool, optional*) – Display version information (default: False)
- **verbose** (*str*) – Set verbosity of runtime messages (default: v) v: verbose q: quiet
- **keep_data_err** (*bool*) – If True, errored directory where simulation
- **is** (*occurred*) – kept.
- **include** (*str, optional*) – List input files that need to be copied to the simulation directory. If a string is provided, it should be in a glob form (see `pathlib.Path.glob()`).

- **process_files** (*bool*) – If True, process the output files and load to a `DataFrame`. Custom processes can be passed using the `custom_processes` attribute.
- **custom_processes** (*dict*(*Callback*)) – if provided, it has to be a dictionary with the keys being a glob (see `pathlib.Path.glob()`), and the value a Callback taking as signature `callback(file: str, working_dir, simulname) -> Any`. All the file matching this glob will be processed by this callback. Note: they will still be processed by `pandas.read_csv` (if they are csv files), resulting in duplicate. The only way to bypass this behavior is to add the key `*.csv` to that dictionary.
- **return_idf** (*bool*) – If True, returns the `IDF` object part of the return tuple.
- **return_files** (*bool*) – If True, all files paths created by the EnergyPlus run are returned.

Raises `EnergyPlusProcessError` – If an issue occurs with the execution of the energyplus command.

See also:

`IDF.simulation_files`, `IDF.processed_results` for simulation outputs.

savecopy(*filename*, *lineendings*='default', *encoding*='latin-1')

Save a copy of the file with the filename passed.

Parameters

- **filename** (*str*) – Filepath to save the file.
- **lineendings** (*str*) – Line endings to use in the saved file. Options are 'default', 'windows' and 'unix' the default is 'default' which uses the line endings for the current system.
- **encoding** (*str*) – Encoding to use for the saved file. The default is 'latin-1' which is compatible with the EnergyPlus IDFEditor.

Returns The new file path.

Return type Path

copy()

Return a copy of self as an in memory IDF.

The copy is a new IDF object with the same parameters and arguments as self but is not attached to an file. Use `IDF.saveas("idfname.idf", inplace=True)` to save the copy to a file inplace. `self.idfname` will now be `idfname.idf`

save(*lineendings*='default', *encoding*='latin-1', ***kwargs*)

Write the IDF model to the text file.

Uses `save()` but also brings over existing simulation results.

Parameters

- **lineendings** (*str*) – Line endings to use in the saved file. Options are 'default', 'windows' and 'unix' the default is 'default' which uses the line endings for the current system.
- **encoding** (*str*) – Encoding to use for the saved file. The default is 'latin-1' which is compatible with the EnergyPlus IDFEditor.

Returns The IDF model

Return type `IDF`

saveas(*filename*, *lineendings*='default', *encoding*='latin-1', *inplace*=False)

Save the IDF model as.

Writes a new text file and load a new instance of the IDF class (new object).

Parameters

- **filename** (*str*) – Filepath to save the file. If None then use the `IDF.idfname` parameter. Also accepts a file handle.
- **lineendings** (*str*) – Line endings to use in the saved file. Options are ‘default’, ‘windows’ and ‘unix’ the default is ‘default’ which uses the line endings for the current system.
- **encoding** (*str*) – Encoding to use for the saved file. The default is ‘latin-1’ which is compatible with the EnergyPlus IDFEditor.
- **inplace** (*bool*) – If True, applies the new filename to self directly, else a new object is returned with the new filename.

Returns A new IDF object based on the new location file.

Return type *IDF*

process_results()

Return the list of processed results.

Processes are defined by *custom_processes* as a list of tuple(file, result). A default process looks for csv files and tries to parse them into *DataFrame* objects.

Returns List of two-tuples.

Return type *list*

Info: For `process_results` to work more consistently, it may be necessary to add the “readvars=True” parameter to *IDF.simulate()* as this one is set to false by default.

add_idf_object_from_idf_string(idf_string)

Add an IDF object (or more than one) from an EnergyPlus text string.

Parameters **idf_string** (*str*) – A text string fully describing an EnergyPlus object.

upgrade(to_version=None, overwrite=True)

EnergyPlus idf version updater using local transition program.

Update the EnergyPlus simulation file (.idf) to the latest available EnergyPlus version installed on this machine. Optionally specify a version (eg.: “9-2-0”) to aim for a specific version. The output will be the path of the updated file. The run is `multiprocessing_safe`.

Hint: If attempting to upgrade an earlier version of EnergyPlus (pre-v7.2.0), specific binaries need to be downloaded and copied to the `EnergyPlus*/PreProcess/IDFVersionUpdater` folder. More info at [Converting older version files](#).

Parameters

- **to_version** (*str*, *optional*) – EnergyPlus version in the form “X-X-X”.
- **overwrite** (*bool*) – If True, original idf file is overwritten with new transitioned file.

Raises

- **EnergyPlusProcessError** – If version updater fails.
- **EnergyPlusVersionError** –
- **CalledProcessError** –

wwr(*azimuth_threshold=10, round_to=10*)

Return the Window-to-Wall Ratio by major orientation.

Optionally round up the WWR value to nearest value (eg.: nearest 10 %).

Parameters

- **azimuth_threshold** (*int*) – Defines the incremental major orientation azimuth angle. Due to possible rounding errors, some surface azimuth can be rounded to values different than the main directions (eg.: 89 degrees instead of 90 degrees). Defaults to increments of 10 degrees.
- **round_to** (*float*) – Optionally round the WWR value to nearest value (eg.: nearest 10). If None, this is ignored and the float is returned.

Returns A DataFrame with the total wall area, total window area and WWR for each main orientation of the building.

Return type (pd.DataFrame)

space_heating_profile(*units='kWh', energy_out_variable_name=None, name='Space Heating', EnergySeries_kwds=None*)

Return space-heating time series.

Parameters

- **units** (*str*) – Units to convert the energy profile to. Will detect the units of the EnergyPlus results.
- **energy_out_variable_name** (*list-like*) – a list of EnergyPlus Variable names.
- **name** (*str*) – Name given to the EnergySeries.
- **EnergySeries_kwds** (*dict, optional*) – keywords passed to EnergySeries.from_sqlite()

Returns EnergySeries

space_cooling_profile(*units='kWh', energy_out_variable_name=None, name='Space Cooling', EnergySeries_kwds=None*)

Return space-cooling time series.

Parameters

- **units** (*str*) – Units to convert the energy profile to. Will detect the units of the EnergyPlus results.
- **energy_out_variable_name** (*list-like*) – a list of EnergyPlus
- **name** (*str*) – Name given to the EnergySeries.
- **EnergySeries_kwds** (*dict, optional*) – keywords passed to EnergySeries.from_sqlite()

Returns EnergySeries

service_water_heating_profile(*units='kWh', energy_out_variable_name=None, name='Space Heating', EnergySeries_kwds=None*)

Return service water heating (domestic hot water) time series.

Parameters

- **units** (*str*) – Units to convert the energy profile to. Will detect the units of the EnergyPlus results.
- **energy_out_variable_name** (*list-like*) – a list of EnergyPlus Variable names.

- **name** (*str*) – Name given to the EnergySeries.
- **EnergySeries_kwds** (*dict, optional*) – keywords passed to EnergySeries.
from_sqlite()

Returns EnergySeries

custom_profile(*energy_out_variable_name, name, units='kWh', prep_outputs=None, EnergySeries_kwds=None*)

Return user-defined time series.

Parameters

- **energy_out_variable_name** (*list-like*) – a list of EnergyPlus
- **name** (*str*) – Name given to the EnergySeries.
- **units** (*str*) – Units to convert the energy profile to. Will detect the units of the EnergyPlus results.
- **prep_outputs** –
- **EnergySeries_kwds** (*dict, optional*) – keywords passed to EnergySeries.
from_sqlite()

Returns EnergySeries

newidfobject(*key, **kwargs*) → Optional[geomeppy.patches.EpBunch]

Define EpBunch object and add to model.

The function will test if the object exists to prevent duplicates.

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- ****kwargs** – Keyword arguments in the format *field=value* used to set fields in the Energy-Plus object.

Example

```
>>> from archetypal import IDF
>>> IDF.newidfobject(
>>>     key="Schedule:Constant".upper(),
>>>     Name="AlwaysOn",
>>>     Schedule_Type_Limits_Name="",
>>>     Hourly_Value=1,
>>> )
```

Returns the object, if successful None: If an error occurred.

Return type EpBunch

addidfobject(*new_object*) → geomeppy.patches.EpBunch

Add an IDF object to the model.

Parameters **new_object** (*EpBunch*) – The IDF object to add.

Returns object.

Return type EpBunch

removeidfobject(*idfobject*)

Remove an IDF object from the model.

Parameters **idfobject** (*EpBunch*) – The object to remove from the model.

removeidfobjects(*idfobjects: Iterable[geomeppy.patches.EpBunch]*)

Remove an IDF object from the model.

Parameters **idfobjects** – The object to remove from the model.

anidfobject(*key: str, aname: str = "", **kwargs*) → *geomeppy.patches.EpBunch*

Define and create an object, but don't add it to the model.

See `newidfobject()`. If you don't specify a value for a field, the default value will be set.

Example

```
>>> from archetypal import IDF
>>> IDF.anidfobject("CONSTRUCTION")
>>> IDF.anidfobject(
>>>     key="CONSTRUCTION",
>>>     Name='Interior Ceiling_class',
>>>     Outside_Layer='LW Concrete',
>>>     Layer_2='soundmat'
>>> )
```

Parameters

- **key** (*str*) – The type of IDF object. This must be in ALL_CAPS.
- **aname** (*str*) – This parameter is not used. It is left there for backward compatibility.
- **kwargs** – Keyword arguments in the format *field=value* used to set fields in the EnergyPlus object.

Returns object.

Return type *EpBunch*

get_schedule_type_limits_data_by_name(*schedule_limit_name*)

Return the data for a particular 'ScheduleTypeLimits' object.

Parameters **schedule_limit_name** –

get_schedule_epbunch(*name, sch_type=None*)

Return the epbunch of a particular schedule name.

If the schedule type is known, retrieve it quicker.

Parameters

- **name** (*str*) – The name of the schedule to retrieve in the IDF file.
- **sch_type** (*str*) – The schedule type, e.g.: "SCHEDULE:YEAR".

property width

Get the width of the building [m].

property length

Get the length of the building [m].

resize(*width*, *length*)

Resize the floor plate to x and y [meters].

Parameters

- **width** (*float*) – The new width [m] of the building (x axis).
- **length** (*float*) – The new length [m] of the building (y axis).

rename(*objkey*, *objname*, *newname*)

Rename all the references to this objname.

Function comes from eppy.modeleditor and was modified to compare the name to rename with a lower string (see `idfobject[idfobject.objls[findex]].lower() == objname.lower()`).

Parameters

- **objkey** (*str*) – EpBunch we want to rename and rename all the occurrences where this object is in the IDF file
- **objname** (*str*) – The name of the EpBunch to rename
- **newname** (*str*) – New name used to rename the EpBunch

Returns The renamed idf object

Return type theobject (EpBunch)

set_wwr(*wwr*: *Optional[float] = None*, *construction*: *Optional[str] = None*, *force*: *bool = False*, *wwr_map*: *Optional[dict] = None*, *orientation*: *Optional[str] = None*, *surfaces*: *Optional[Iterable] = None*)

Set Window-to-Wall Ratio of external walls.

Different WWR can be applied to specific wall orientations using the *wwr_map* dictionary. This map is a dict of wwr values, keyed by *wall.azimuth*, which overrides the default passed as *wwr*. Note that *wall.azimuth* is rounded to the closest integer.

They can also be applied to walls oriented to a compass point, e.g. north, which will apply to walls which have an azimuth within 45 degrees of due north.

Parameters

- **wwr** – The window to wall ratio to apply to all orientations. If *wwr_map* is specified, *wwr* is the default wwr for oreintations not defined in the mapping.
- **construction** – Name of a window construction to apply.
- **force** – True to create windows on walls that don't have any.
- **wwr_map** – Mapping from wall orientation (azimuth) to WWR, e.g. {180: 0.25, 90: 0.2}.
- **orientation** – One of "north", "east", "south", "west". Walls within 45 degrees will be affected.
- **surfaces** – Iterable of surfaces to set the window to wall ratio of.

archetypal.idfclass.Outputs

class archetypal.idfclass.Outputs(idf)

Handles preparation of EnergyPlus outputs. Different instance methods allow to chain methods together and to add predefined bundles of outputs in one go.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF(prepare_outputs=False) # True by default
>>> idf.outputs.add_output_control().add_umi_outputs(
>>> ).add_profile_gas_elect_outputs().apply()
```

Initialize an outputs object.

Parameters **idf** (*IDF*) – the IDF object for which this outputs object is created.

add_custom(outputs)

Add custom-defined outputs as a list of objects.

Examples

```
>>> outputs = IDF().outputs
>>> to_add = dict(
>>>     key= "OUTPUT:METER",
>>>     Key_Name="Electricity:Facility",
>>>     Reporting_Frequency="hourly",
>>> )
>>> outputs.add_custom([to_add]).apply()
```

Parameters **outputs** (*list*, *bool*) – Pass a list of ep-objects defined as dictionary. See examples. If a bool, ignored.

Returns self

Return type *Outputs*

add_basics()

Adds the summary report and the sql file to the idf outputs

add_schedules()

Adds Schedules object

add_meter_variables(format='IDF')

Generate .mdd file at end of simulation. This file (from the Output:VariableDictionary, regular; and Output:VariableDictionary, IDF; commands) shows all the report meters along with their “availability” for the current input file. A user must first run the simulation (at least semi-successfully) before the available output meters are known. This output file is available in two flavors: regular (listed as they are in the Input Output Reference) and IDF (ready to be copied and pasted into your Input File).

Parameters **format** (*str*) – Choices are “IDF” and “regular”

Returns self

Return type *Outputs*

add_summary_report(*summary*='AllSummary')

Adds the `Output:Table:SummaryReports` object.

Parameters *summary* (*str*) – Choices are AllSummary, AllMonthly, AllSummaryAndMonthly, AllSummaryAndSizingPeriod, AllSummaryMonthlyAndSizingPeriod, Annual-BuildingUtilityPerformanceSummary, InputVerificationandResultsSummary, SourceEnergyEndUseComponentsSummary, ClimaticDataSummary, EnvelopeSummary, SurfaceShadingSummary, ShadingSummary, LightingSummary, EquipmentSummary, HVACSizingSummary, ComponentSizingSummary, CoilSizingDetails, OutdoorAirSummary, SystemSummary, AdaptiveComfortSummary, SensibleHeatGainSummary, Standard62.1Summary, EnergyMeters, InitializationSummary, LEEDSummary, TariffReport, EconomicResultSummary, ComponentCostEconomicsSummary, LifeCycleCostReport, HeatEmissionsSummary,

Returns self

Return type *Outputs*

add_sql(*sql_output_style*='SimpleAndTabular')

Adds the `Output:SQLite` object. This object will produce an sql file that contains the simulation results in a database format. See [eplusout.sql](#) for more details.

Parameters *sql_output_style* (*str*) – The *Simple* option will include all of the predefined database tables as well as time series related data. Using the *SimpleAndTabular* choice adds database tables related to the tabular reports that are already output by EnergyPlus in other formats.

Returns self

Return type *Outputs*

add_output_control(*output_control_table_style*='CommaAndHTML')

Sets the `OutputControl:Table:Style` object.

Parameters *output_control_table_style* (*str*) – Choices are: Comma, Tab, Fixed, HTML, XML, CommaAndHTML, TabAndHTML, XMLAndHTML, All

Returns self

Return type *Outputs*

add_umi_template_outputs()

Adds the necessary outputs in order to create an UMI template.

add_umi_ouputs()

Adds the necessary outputs in order to return the same energy profile as in UMI.

add_profile_gas_elect_ouputs()

Adds the following meters: Electricity:Facility, Gas:Facility, WaterSystems:Electricity, Heating:Electricity, Cooling:Electricity

apply()

Applies the outputs to the idf model. Modifies the model by calling `newidfobject()`

archetypal.idfclass.Meters

class archetypal.idfclass.**Meters**(idf)

Lists available meters in the IDF model.

Once simulated at least once, the IDF.meters attribute is populated with meters categories (“Output:Meter” or “Output:Meter:Cumulative”) and each category is populated with all the available meters.

Example

For example, to retrieve the WaterSystems:MainsWater meter, simply call

```
>>> from archetypal import IDF
>>> idf = IDF() # load an actual idf file here
>>> idf.meters.OutputMeter.WaterSystems__MainsWater.values()
```

Hint: Available meters are read from the .mdd file

Initialize Meter.

OutputMeter

alias of archetypal.idfclass.meters.MeterGroup

archetypal.idfclass.Variables

class archetypal.idfclass.**Variables**(idf)

Lists available variables in the IDF model.

Once simulated at least once, the IDF.variables attribute is populated with variable categories and each category is populated with all the available variables.

Example

For example, to retrieve the “Zone Operative Temperature” variable, simply call

```
>>> from archetypal import IDF
>>> idf = IDF() # load an actual idf file here
>>> idf.variables.OutputVariable.Zone_Operative_Temperature.values()
```

Hint: Available meters are read from the .mdd file

Initialize MeterGroup.

OutputVariable

alias of archetypal.idfclass.variables.VariableGroup

1.12.2 UMI Template Library

*UmiTemplateLibrary*Handles parsing and creating Template Library Files for UMI for Rhino.

archetypal.umi_template.UmiTemplateLibrary

```
class archetypal.umi_template.UmiTemplateLibrary(name='unnamed', BuildingTemplates=None,
                                                GasMaterials=None, GlazingMaterials=None,
                                                OpaqueConstructions=None,
                                                OpaqueMaterials=None,
                                                WindowConstructions=None,
                                                StructureInformations=None, DaySchedules=None,
                                                WeekSchedules=None, YearSchedules=None,
                                                DomesticHotWaterSettings=None,
                                                VentilationSettings=None, WindowSettings=None,
                                                ZoneConditionings=None,
                                                ZoneConstructionSets=None, ZoneLoads=None,
                                                ZoneDefinitions=None)
```

Handles parsing and creating Template Library Files for UMI for Rhino.

- See [open\(\)](#) to parse existing Umi Template Library files (.json).
- See [from_idf_files\(\)](#) to create a library by converting existing IDF models.

Initialize a new UmiTemplateLibrary with empty attributes.

Parameters

- **name** (*str*) – The name of the UMI Template.
- **BuildingTemplates** (*list of BuildingTemplate*) – list of BuildingTemplate objects.
- **GasMaterials** (*list of GasMaterial*) – list of GasMaterial objects.
- **GlazingMaterials** (*list of GlazingMaterial*) – list of GlazingMaterial objects.
- **OpaqueConstructions** (*list of OpaqueConstruction*) – list of OpaqueConstruction objects.
- **OpaqueMaterials** (*list of OpaqueMaterial*) – list of OpaqueMaterial objects.
- **WindowConstructions** (*list of WindowConstruction*) – list of WindowConstruction objects.
- **StructureInformations** (*list of StructureInformation*) – list of StructureInformation objects.
- **DaySchedules** (*list of DaySchedule*) – list of DaySchedule objects.
- **WeekSchedules** (*list of WeekSchedule*) – list of WeekSchedule objects.
- **YearSchedules** (*list of YearSchedule*) – list of YearSchedule objects.
- **DomesticHotWaterSettings** (*list of DomesticHotWaterSetting*) – list of DomesticHotWaterSetting objects.
- **VentilationSettings** (*list of VentilationSetting*) – list of VentilationSetting objects.
- **WindowSettings** (*list of WindowSetting*) – list of WindowSetting objects.

- **ZoneConditionings** (*list of ZoneConditioning*) – list of ZoneConditioning objects.
- **ZoneConstructionSets** (*list of ZoneConstructionSet*) – list of ZoneConstructionSet objects.
- **ZoneLoads** (*list of ZoneLoad*) – list of ZoneLoad objects.
- **ZoneDefinitions** (*list of ZoneDefinition*) – list of Zone objects

classmethod `from_idf_files(idf_files, weather, name='unnamed', processors=-1, keep_all_zones=False, **kwargs)`

Initialize an UmiTemplateLibrary object from one or more idf_files.

The resulting object contains the reduced version of the IDF files. To save to file, call the `save()` method.

Important: When using `from_idf_files()` The idf files are striped of run period modifiers and special days to return simple annual schedules.

Parameters

- **idf_files** (*list of (str or Path)*) – list of IDF file paths.
- **weather** (*str or Path*) – Path to the weather file.
- **name** (*str*) – The name of the Template File
- **processors** (*int*) – Number of cores. Defaults to -1, all cores.
- **kwargs** – keyword arguments passed to IDF().

Raises **Exception** – All exceptions are raised if settings.debug=True. Will raise an exception if all BuildingTemplates failed to be created.

static `template_complexity_reduction(idfname, epw, **kwargs)`

Wrap IDF, simulate and BuildingTemplate for parallel processing.

classmethod `open(filename)`

Initialize an UmiTemplate object from an UMI Template Library File.

Parameters **filename** (*str or Path*) – PathLike object giving the pathname of the UMI Template File.

Returns The template object.

Return type *UmiTemplateLibrary*

classmethod `loads(s, name)`

load string.

validate (*defaults=True*)

Validate the object.

save (*path_or_buf=None, indent=2, sort_keys=False, compression='infer', storage_options=None*)

Save to json file.

Parameters

- **path_or_buf** (*path-like*) – File path or object. If not specified, overwrites files. See `UmiTemplateLibrary.name`.
- **indent** (*bool or str or int*) – If indent is a non-negative integer or string, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0, negative, or "" will only insert newlines. None (the default) selects the most

compact representation. Using a positive integer indent indents that many spaces per level. If indent is a string (such as “t”), that string is used to indent each level.

- **sort_keys** (*callable*) – If sort_keys is true (default: False), then the output of dictionaries will be sorted by this callable. e.g.: `lambda x: x.get(“$id”)` sorts by \$id. If callable is not available or fails, then sorted by *Name*.
- **compression** (*str*) – A string representing the compression to use in the output file, only used when the first argument is a filename. By default, the compression is inferred from the filename.
- **storage_options** (*dict*) – Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc., if using a URL that will be parsed by fsspec, e.g., starting “s3://”, “gcs://”. An error will be raised if providing this argument with a non-fsspec URL. See the fsspec and backend storage implementation docs for the set of allowed keys and values.

to_json(*path_or_buf=None, indent=2, sort_keys=False, default_handler=None, compression='infer', storage_options=None*)

Convert the object to a JSON string.

Parameters

- **path_or_buf** (*path-like*) – File path or object. If not specified, the result is returned as a string.
- **indent** (*bool or str or int*) – If indent is a non-negative integer or string, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0, negative, or “” will only insert newlines. None (the default) selects the most compact representation. Using a positive integer indent indents that many spaces per level. If indent is a string (such as “t”), that string is used to indent each level.
- **sort_keys** (*callable*) –
- **default_handler** (*callable*) – Handler to call if object cannot otherwise be converted to a suitable format for JSON. Should receive a single argument which is the object to convert and return a serializable object.
- **compression** (*str*) – A string representing the compression to use in the output file, only used when the first argument is a filename. By default, the compression is inferred from the filename.
- **storage_options** (*dict*) – Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc., if using a URL that will be parsed by fsspec, e.g., starting “s3://”, “gcs://”. An error will be raised if providing this argument with a non-fsspec URL. See the fsspec and backend storage implementation docs for the set of allowed keys and values.

to_dict()

Return UmiTemplateLibrary dictionary representation.

unique_components(*exceptions=None*)

Keep only unique components.

Calls `get_unique()` for each object in the graph.

replace_component(*this, that*) → *None*

Replace all instances of *this* with *that*.

Parameters

- **this** (*UmiBase*) – The reference to replace with *that*.

- **that** (*UmiBase*) – The object to replace each references with.

update_components_list(*exceptions=None*)

Update the component groups with connected components.

build_graph()

Create the `networkx.DiGraph` UmiBase objects as nodes.

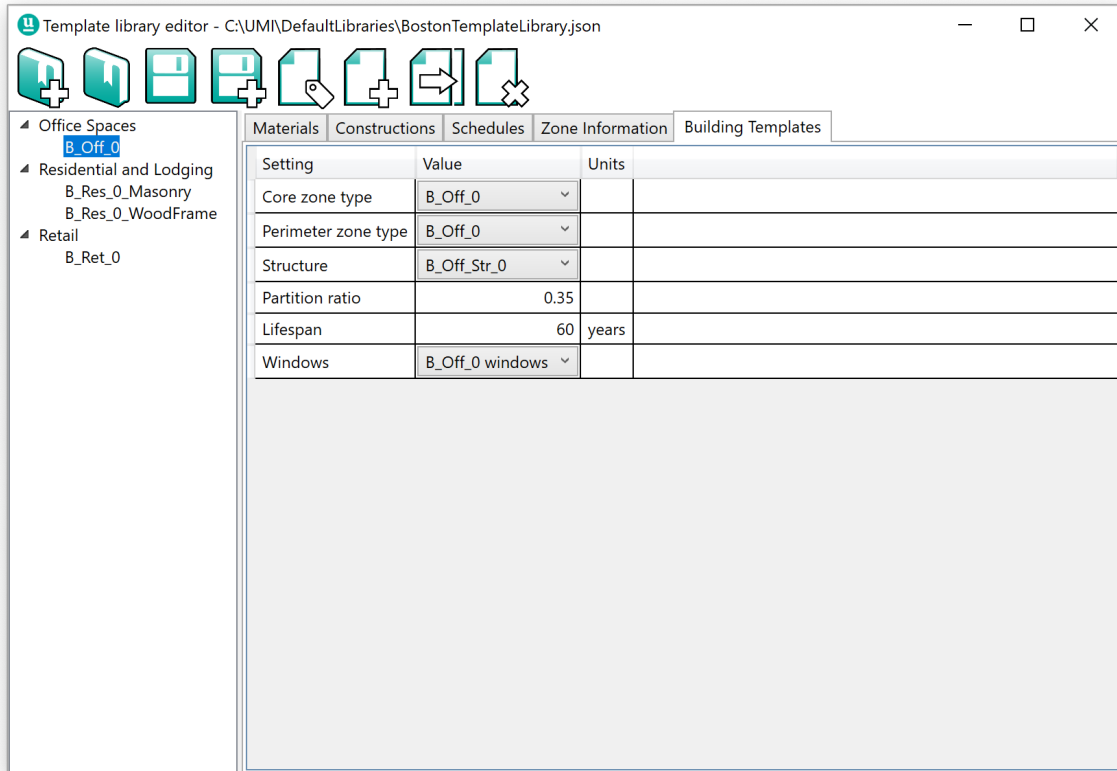
1.12.3 Template Classes

<i>BuildingTemplate</i>	Main class supporting the definition of a single building template.
<i>ZoneConditioning</i>	HVAC settings for the zone.
<i>DomesticHotWaterSetting</i>	Domestic Hot Water settings.
<i>GasMaterial</i>	Gas Materials.
<i>GlazingMaterial</i>	Glazing Materials class.
<i>ZoneLoad</i>	Zone Loads.
<i>OpaqueConstruction</i>	Opaque Constructions.
<i>OpaqueMaterial</i>	Use this component to create a custom opaque material.
<i>UmiSchedule</i>	Class that handles Schedules.
<i>StructureInformation</i>	Building Structure settings.
<i>VentilationSetting</i>	Zone Ventilation Settings.
<i>WindowConstruction</i>	Window Construction.
<i>WindowSetting</i>	Defines the various window-related properties of a Zone.
<i>ZoneDefinition</i>	Zone settings class.
<i>ZoneConstructionSet</i>	ZoneConstructionSet class.

archetypal.template.BuildingTemplate

```
class archetypal.template.BuildingTemplate(Name, Core=None, Perimeter=None, Structure=None,
                                           Windows=None, Lifespan=60, PartitionRatio=0.35,
                                           DefaultWindowToWallRatio=0.4, YearFrom=None,
                                           YearTo=None, Country=None, ClimateZone=None,
                                           Authors=None, AuthorEmails=None, Version='v1.0',
                                           **kwargs)
```

Main class supporting the definition of a single building template.



Initialize a *BuildingTemplate* object with the following attributes:

Parameters

- **Core** (*ZoneDefinition*) – The Zone object defining the core zone. see Zone for more details.
- **Perimeter** (*ZoneDefinition*) – The Zone object defining the perimeter zone. see Zone for more details.
- **Structure** (*StructureInformation*) – The StructureInformation object defining the structural properties of the template.
- **Windows** (*WindowSetting*) – The WindowSetting object defining the window properties of the object.
- **Lifespan** (*float*) – The projected lifespan of the building template in years. Used in various calculations such as embodied energy.
- **PartitionRatio** (*float*) – The number of lineal meters of partitions (Floor to ceiling) present in average in the building floor plan by m2.
- **DefaultWindowToWallRatio** (*float*) – The default Window to Wall Ratio (WWR) for this template (same for all orientations). Number between 0 and 1.
- **YearFrom** (*int*) – Start year for range.
- **YearTo** (*int*) – End year for range.
- **Country** (*list of str*) – alpha-3 Country Code.
- **ClimateZone** (*list of str*) – ANSI/ASHRAE/IESNA Standard 90.1 International Climatic Zone. eg. “5A”

- **Authors** (*list of str*) – Authors of this template
- **AuthorEmails** (*list of str*) – Contact information.
- **Version** (*str*) – Version number.
- ****kwargs** – other optional keywords passed to other constructors.

property Perimeter

Get or set the perimeter ZoneDefinition.

property Core

Get or set the core ZoneDefinition.

property Structure

Get or set the StructureInformation.

property Windows

Get or set the WindowSetting.

property DefaultWindowToWallRatio

Get or set the DefaultWindowToWallRatio [-].

property Lifespan

Get or set the building life span [years].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

property PartitionRatio

Get or set the partition ratio [-].

property YearFrom

Get or set the YearFrom [int].

property YearTo

Get or set the YearTo [int].

property Country

Get or set the list of alpha-3 country codes [list].

property ClimateZone

Get or set the list of climatic zones [list].

property Authors

Get or set the list of authors [list].

property AuthorEmails

Get or set list of author emails [list].

property Version

Get or set the template version [str].

classmethod from_dict(*data*, *zone_definitions*, *structure_definitions*, *window_settings*, *schedules*,
window_constructions, ***kwargs*)

Create an BuildingTemplate from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **zone_definitions** (*dict*) – A dictionary of ZoneDefinition objects with their id as keys.
- **structure_definitions** (*dict*) – A dictionary of StructureInformation with their id as keys.
- **window_settings** (*dict*) – A dictionary of WindowSetting objects with their id as keys.
- **schedules** (*dict*) – A dictionary of UmiSchedule with their id as keys.
- **window_constructions** (*dict*) – A dictionary of WindowConstruction objects with their id as keys.
- ****kwargs** – keywords passed to the constructor.

```
{
  "Core": {
    "$ref": "178"
  },
  "Lifespan": 60,
```

(continues on next page)

(continued from previous page)

```

"PartitionRatio": 0.3,
"Perimeter": {
  "$ref": "178"
},
"Structure": {
  "$ref": "64"
},
"Windows": {
  "$ref": "181"
},
"DefaultWindowToWallRatio": 0.4,
"YearFrom": 0,
"YearTo": 0,
"Country": [
  "USA"
],
"ClimateZone": [
  "5A"
],
"Authors": [
  "Carlos Cerezo"
],
"AuthorEmails": [
  "ccerezo@mit.edu"
],
"Version": "v1.0",
"Category": "Residential and Lodging",
"Comments": "Base building definition for MIT 4433",
"DataSource": "MIT_SDL",
"Name": "B_Res_0_WoodFrame"
}

```

classmethod `from_idf(idf, **kwargs)`

Create a BuildingTemplate from an IDF object.

Parameters

- **idf** (*IDF*) –
- ****kwargs** –

classmethod `reduced_model(name, zones, **kwargs)`

Create reduced BuildingTemplate from list of ZoneDefinitions.

Parameters

- **name** (*str*) – The name of the building template.
- **zones** (*list of ZoneDefinition*) – A list of zone definition objects to reduce. At least one must be a perimeter zone (`ZoneDefinition.is_core` is `False`).
- ****kwargs** – keywords passed to the class constructor.

Returns The reduced BuildingTemplate.

Return type *BuildingTemplate*

to_dict()

Return BuildingTemplate dictionary representation.

validate()

Validate object and fill in missing values.

get_unique()

Replace recursively every objects with the first equivalent object.

mapping(validate=True)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

get_ref(ref)

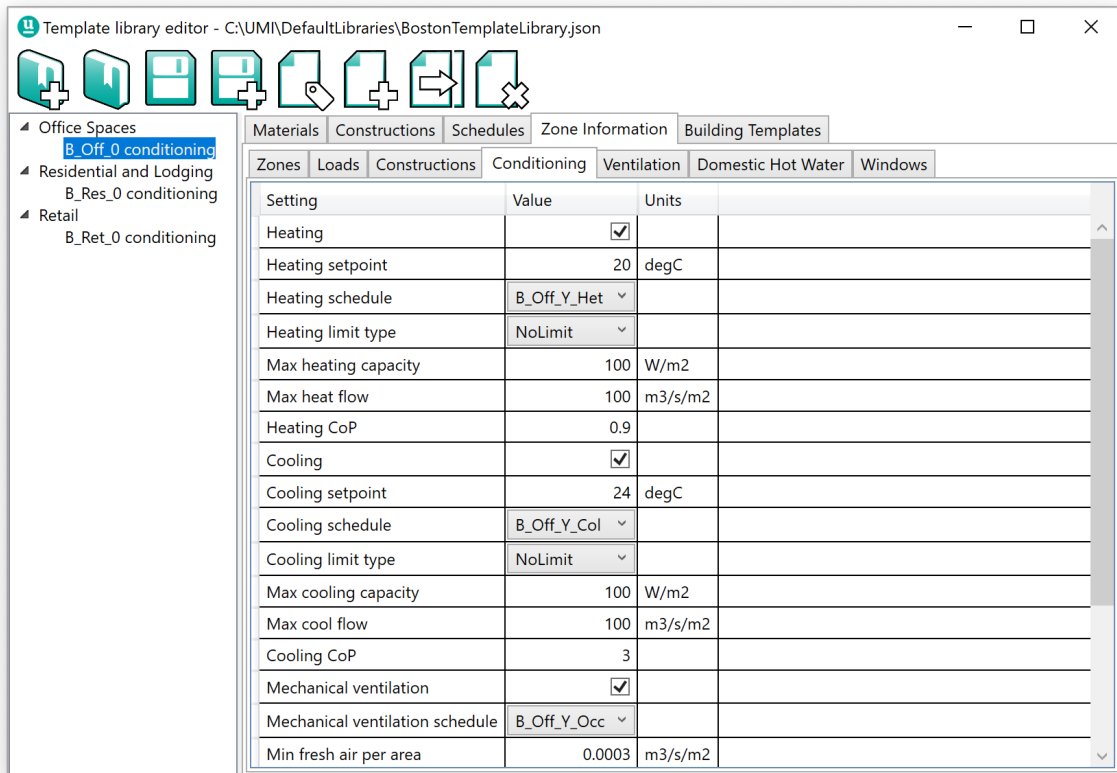
Get item matching reference id.

Parameters **ref** –

archetypal.template.ZoneConditioning

```
class archetypal.template.ZoneConditioning(Name, IsHeatingOn=False, HeatingSetpoint=20,
                                           HeatingSchedule=None,
                                           HeatingLimitType=IdealSystemLimit.NoLimit,
                                           HeatingFuelType=FuelType.NaturalGas,
                                           MaxHeatingCapacity=100, MaxHeatFlow=100,
                                           HeatingCoeffOfPerf=1, IsCoolingOn=False,
                                           CoolingSetpoint=26, CoolingSchedule=None,
                                           CoolingLimitType=IdealSystemLimit.NoLimit,
                                           CoolingFuelType=FuelType.Electricity,
                                           MaxCoolingCapacity=100, MaxCoolFlow=100,
                                           CoolingCoeffOfPerf=1, IsMechVentOn=False,
                                           EconomizerType=EconomizerTypes.NoEconomizer,
                                           MechVentSchedule=None, MinFreshAirPerArea=0,
                                           MinFreshAirPerPerson=0,
                                           HeatRecoveryType=HeatRecoveryTypes.NONE,
                                           HeatRecoveryEfficiencyLatent=0.65,
                                           HeatRecoveryEfficiencySensible=0.7, area=1, **kwargs)
```

HVAC settings for the zone.



Initialize a new *ZoneConditioning* object.

Parameters

- **Name** (*str*) – Name of the object. Must be Unique.
- **IsHeatingOn** (*bool*) – Whether or not heating is available.
- **HeatingSetpoint** (*float*) – The temperature below which zone heating is turned on. Here, we take the mean value over the ye
- **HeatingSchedule** (*UmiSchedule*) – The availability schedule for space heating in this zone. If the value is 0, heating is not available, and heating is not supplied to the zone.
- **HeatingLimitType** (*int*) – The input must be either LimitFlowRate = 1, LimitCapacity = 2, LimitFlowRateAndCapacity = 3 or NoLimit = 0.
- **MaxHeatingCapacity** (*float*) – The maximum allowed sensible heating capacity in Watts if Heating Limit is set to LimitCapacity or LimitFlowRateAndCapacity
- **MaxHeatFlow** (*float*) – The maximum heating supply air flow rate in cubic meters per second if heating limit is set to LimitFlowRate or LimitFlowRateAndCapacity
- **HeatingCoeffOfPerf** (*float*) – Efficiency of heating system. The COP is of each zone is equal, and refer to the COP of the entire building.
- **IsCoolingOn** (*bool*) – Whether or not cooling is available.
- **CoolingSetpoint** (*float*) – The temperature above which the zone heating is turned on. Here, we take the mean value over the ye
- **CoolingSchedule** (*UmiSchedule*) – The availability schedule for space cooling in this zone. If the value is 0, cooling is not available, and cooling is not supplied to the zone.

- **CoolingLimitType** (*str*) – The input must be either LimitFlowRate = 1, LimitCapacity = 2, LimitFlowRateAndCapacity = 3 or NoLimit = 0.
- **MaxCoolingCapacity** (*float*) – The maximum allowed total (sensible plus latent) cooling capacity in Watts per square meter.
- **MaxCoolFlow** (*float*) – The maximum cooling supply air flow rate in cubic meters per second if Cooling Limit is set to LimitFlowRate or LimitFlowRateAndCapacity
- **CoolingCoeffOfPerf** (*float*) – Performance factor of the cooling system. This value is used to calculate the total cooling energy use by dividing the cooling load by the COP. The COP of the zone shared with all zones and refers to the COP of the entire building.
- **IsMechVentOn** (*bool*) – If True, an outdoor air quantity for use by the model is calculated.
- **EconomizerType** (*int*) – Specifies if there is an outdoor air economizer. The choices are: NoEconomizer = 0, DifferentialDryBulb = 1, or DifferentialEnthalpy = 2. For the moment, the EconomizerType is applied for the entire building (every zone with the same EconomizerType). Moreover, since UMI does not support all Economizer Types, some assumptions are made:
 - If ‘NoEconomizer’ in EnergyPlus, EconomizerType=‘NoEconomizer’
 - If ‘DifferentialEnthalpy’ in EnergyPlus, EconomizerType = ‘DifferentialEnthalpy’
 - If ‘DifferentialDryBulb’ in EnergyPlus, EconomizerType = ‘DifferentialDryBulb’
 - If ‘FixedDryBulb’ in EnergyPlus, EconomizerType = ‘DifferentialDryBulb’
 - If ‘FixedEnthalpy’ in EnergyPlus, EconomizerType = ‘DifferentialEnthalpy’
 - If ‘ElectronicEnthalpy’ in EnergyPlus, EconomizerType = ‘DifferentialEnthalpy’
 - If ‘FixedDewPointAndDryBulb’ in EnergyPlus, EconomizerType = ‘DifferentialDryBulb’
 - If ‘DifferentialDryBulbAndEnthalpy’ in EnergyPlus, EconomizerType = ‘DifferentialEnthalpy’
- **MechVentSchedule** (*UmiSchedule*) – The availability schedule of the mechanical ventilation. If the value is 0, the mechanical ventilation is not available and air flow is not requested.
- **MinFreshAirPerArea** (*float*) – The design outdoor air volume flow rate per square meter of floor area (units are m3/s-m2). This input is used if Outdoor Air Method is Flow/Area, Sum or Maximum
- **MinFreshAirPerPerson** (*float*) – The design outdoor air volume flow rate per person for this zone in cubic meters per second per person. The default is 0.00944 (20 cfm per person).
- **HeatRecoveryType** (*int*) – Select from None = 0, Sensible = 1, or Enthalpy = 2. None means that there is no heat recovery. Sensible means that there is sensible heat recovery whenever the zone exhaust air temperature is more favorable than the outdoor air temperature. Enthalpy means that there is latent and sensible heat recovery whenever the zone exhaust air enthalpy is more favorable than the outdoor air enthalpy. The default is None
- **HeatRecoveryEfficiencyLatent** (*float*) – The latent heat recovery effectiveness, where effectiveness is defined as the change in supply humidity ratio divided by the difference in entering supply and relief air humidity ratios. The default is 0.65.
 - If the HeatExchanger is an AirToAir FlatPlate, HeatRecoveryEfficiencyLatent = HeatRecoveryEfficiencySensible - 0.05
 - If the HeatExchanger is an AirToAir SensibleAndLatent, we suppose that HeatRecoveryEfficiencyLatent = Latent Effectiveness at 100% Heating Air Flow

- If the HeatExchanger is a Desiccant BalancedFlow, we use the default value for the efficiency (=0.65).
- **HeatRecoveryEfficiencySensible** (*float*) – The sensible heat recovery effectiveness, where effectiveness is defined as the change in supply temperature divided by the difference in entering supply and relief air temperatures. The default is 0.70.
 - If the HeatExchanger is an AirToAir FlatPlate, $\text{HeatRecoveryEfficiencySensible} = (\text{Supply Air Outlet } T^{\circ}\text{C} - \text{Supply Air Inlet } T^{\circ}\text{C}) / (\text{Secondary Air Inlet } T^{\circ}\text{C} - \text{Supply Air Inlet } T^{\circ}\text{C})$
 - If the HeatExchanger is an AirToAir SensibleAndLatent, we suppose that $\text{HeatRecoveryEfficiencySensible} = \text{Sensible Effectiveness at 100\% Heating Air Flow}$
 - If the HeatExchanger is a Desiccant BalancedFlow, we use the default value for the efficiency (=0.70)
- ****kwargs** – Other arguments passed to the base class `archetypal.template.UmiBase`

property area

Get or set the area of the zone associated to this object [m²].

property CoolingSetpoint

Get or set the cooling setpoint [degC].

property HeatingSetpoint

Get or set the heating setpoint [degC].

property MaxCoolFlow

Get or set the maximum cooling flowrate [m³/s/m²].

property MaxHeatFlow

Get or set the maximum heating flowrate [m³/s/m²].

property MaxHeatingCapacity

Get or set the maximum heating capacity [W/m²].

property MaxCoolingCapacity

Get or set the maximum cooling capacity [W/m²].

property MinFreshAirPerArea

Get or set the minimum fresh air per area [m³/s/m²].

property MinFreshAirPerPerson

Get or set the minimum fresh air per person [m³/s/p].

property IsHeatingOn

Get or set the availability of heating [bool].

property HeatingSchedule

Get or set the heating availability schedule.

property HeatingLimitType

Get or set the heating limit type [enum].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property HeatingFuelType

Get or set the heating fuel type [enum].

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights*=None)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

to_ref()

Return a ref pointer to self.

property HeatingCoeffOfPerf

Get or set the heating COP [-].

property IsCoolingOn

Get or set the availability of cooling [bool].

property CoolingSchedule

Get or set the cooling availability schedule.

property CoolingLimitType

Get or set the cooling limit type [enum].

property CoolingFuelType

Get or set the cooling fuel type [enum].

property CoolingCoeffOfPerf

Get or set the cooling COP [-].

property IsMechVentOn

Get or set the availability of mechanical ventilation [bool].

property EconomizerType

Get or set the economizer type [enum].

property MechVentSchedule

Get or set the outdoor air requirements over time.

property HeatRecoveryType

Get or set the heat recovery type.

property HeatRecoveryEfficiencyLatent

Get or set the latent heat recovery effectiveness [-].

property HeatRecoveryEfficiencySensible

Get or set the sensible heat recovery effectiveness [-].

classmethod from_dict(*data*, *schedules*, *kwargs*)**

Create a ZoneConditioning from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **schedules** (*dict*) – A dictionary of UmiSchedules with their id as keys.
- ****kwargs** – keywords passed to parent constructor.

```
{
    "$id": "165",
    "CoolingSchedule": { $ref: "1" },
    "CoolingCoeffOfPerf": 3.0,
    "CoolingSetpoint": 24.0,
    "CoolingLimitType": 0,
    "CoolingFuelType": 1,
    "EconomizerType": 0,
    "HeatingCoeffOfPerf": 0.9,
    "HeatingLimitType": 0,
    "HeatingFuelType": 2,
    "HeatingSchedule": { $ref: "2" },
    "HeatingSetpoint": 20.0,
    "HeatRecoveryEfficiencyLatent": 0.65,
    "HeatRecoveryEfficiencySensible": 0.7,
    "HeatRecoveryType": 0,
    "IsCoolingOn": True,
    "IsHeatingOn": True,
    "IsMechVentOn": True,
    "MaxCoolFlow": 100.0,
    "MaxCoolingCapacity": 100.0,
    "MaxHeatFlow": 100.0,
    "MaxHeatingCapacity": 100.0,
    "MechVentSchedule": { $ref: "3" },
    "MinFreshAirPerArea": 0.0003,
    "MinFreshAirPerPerson": 0.0025,
    "Category": "Office Spaces",
    "Comments": None,
    "DataSource": "MIT_SDL",
}
```

(continues on next page)

(continued from previous page)

```
"Name": "B_Off_0 Conditioning",  
}
```

to_dict()

Return ZoneConditioning dictionary representation.

classmethod from_zone(*zone, zone_ep, nolimit=False, **kwargs*)

Create a ZoneConditioning object from a zone.

Parameters

- **zone_ep** –
- **zone** (*archetypal.template.zone.Zone*) – zone to gets information from.

static get_equipment_list(*zone, zone_ep*)

Get zone equipment list.

Parameters **zone_ep** –**fresh_air_from_ideal_loads**(*zone, zone_ep*)

Resolve fresh air requirements for Ideal Loads Air System.

Parameters

- **zone_ep** –
- **zone** –

Returns (IsMechVentOn, MinFreshAirPerArea, MinFreshAirPerPerson, MechVentSchedule)**Return type** 4-tuple**fresh_air_from_zone_sizes**(*zone*)

Return the Mechanical Ventilation from the ZoneSizes Table in the sql db.

Parameters **zone** (*ZoneDefinition*) –**Returns** (IsMechVentOn, MinFreshAirPerArea, MinFreshAirPerPerson, MechVentSchedule)**Return type** 4-tuple**combine**(*other, weights=None*)

Combine two ZoneConditioning objects together.

Parameters

- **other** (*ZoneConditioning*) – The other ZoneConditioning object to combine with.
- **weights** (*list-like, optional*) – A list-like object of len 2. If None, the volume of the zones for which self and other belongs is used.

Returns the combined ZoneConditioning object.**Return type** (*ZoneConditioning*)**validate()**

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.**to_epbunch**(*idf, zone_name, design_specification_outdoor_air_object*)

Convert self to an EpBunch given an IDF model.

Parameters

- **idf** –
- **zone_name** –

Returns The EpBunch object added to the idf model.

Return type EpBunch

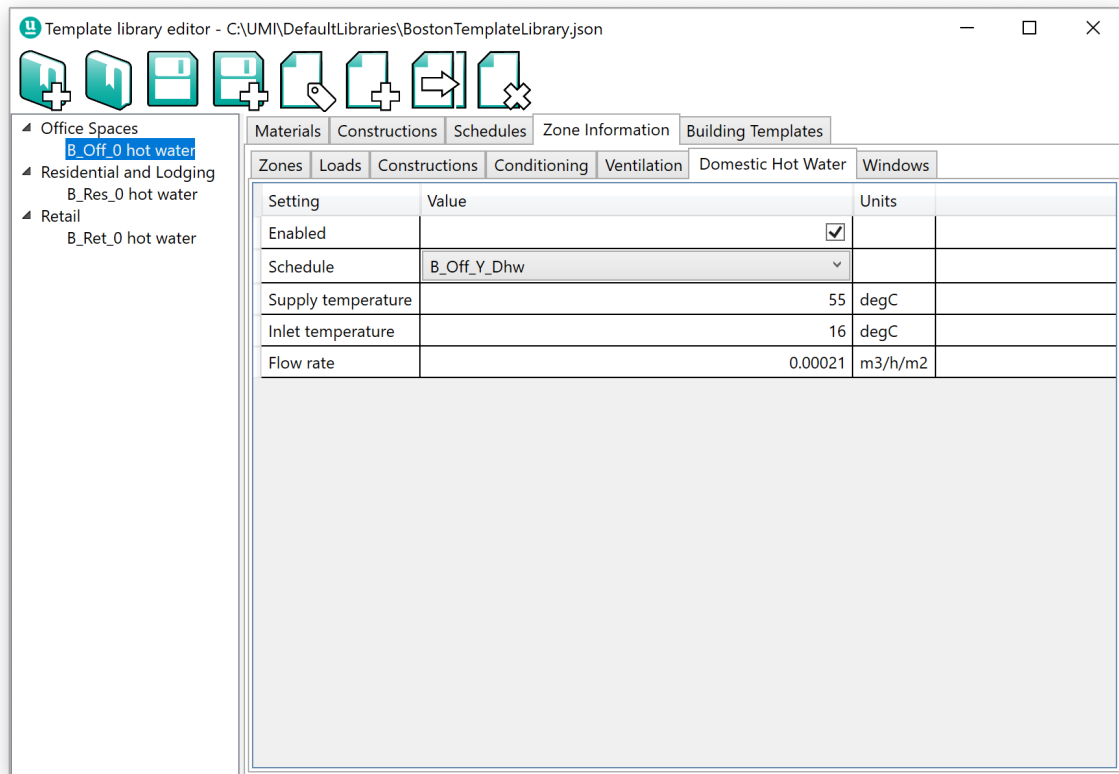
duplicate()

Get copy of self.

archetypal.template.DomesticHotWaterSetting

```
class archetypal.template.DomesticHotWaterSetting(Name, WaterSchedule=None, IsOn=True,
                                                    FlowRatePerFloorArea=0.03,
                                                    WaterSupplyTemperature=65,
                                                    WaterTemperatureInlet=10, area=1, **kwargs)
```

Domestic Hot Water settings.



Initialize object with parameters.

Parameters

- **area** (*float*) – The area the zone associated to this object.
- **IsOn** (*bool*) – If True, dhw is on.
- **WaterSchedule** (*UmiSchedule*) – Schedule that modulates the FlowRatePerFloorArea.
- **FlowRatePerFloorArea** (*float*) – The flow rate per flow area [$\text{m}^3/(\text{hr} \cdot \text{m}^2)$].

- **WaterSupplyTemperature** (*float*) – The water supply temperature [degC].
- **WaterTemperatureInlet** (*float*) – The water temperature intel from the water mains [degC].
- ****kwargs** – keywords passed to parent constructors.

property FlowRatePerFloorArea

Get or set the flow rate per flow area [$\text{m}^3/(\text{hr}\cdot\text{m}^2)$].

property IsOn

Get or set the availability of the domestic hot water [bool].

property WaterSchedule

Get or set the schedule which modulates the FlowRatePerFloorArea.

property WaterSupplyTemperature

Get or set the water supply temperature [degC].

property WaterTemperatureInlet

Get or set the water temperature intel from the water mains [degC].

property area

Get or set the area of the zone associated to this object [m^2].

classmethod from_dict(*data*, *schedules*, ****kwargs**)

Create a DomesticHotWaterSetting from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **schedules** (*dict*) – A dictionary of UmiSchedules with their id as keys.
- ****kwargs** – keywords passed the MaterialBase constructor.

to_dict()

Return DomesticHotWaterSetting dictionary representation.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

combine(*other*, ****kwargs**)

Combine two DomesticHotWaterSetting objects together.

Notes

When combining 2 DomesticHotWater Settings objects, the WaterSchedule must be averaged via the final quantity which is the peak floor rate [m3/hr/m2] * area [m2].

```
(
    np.average(
        [zone_1.WaterSchedule.all_values, zone_2.WaterSchedule.all_values],
        axis=0,
        weights=[
            zone_1.FlowRatePerFloorArea * zone_1.area,
            zone_2.FlowRatePerFloorArea * zone_2.area,
        ],
    )
    * (combined.FlowRatePerFloorArea * 100)
).sum()
```

Parameters

- **other** (*DomesticHotWaterSetting*) – The other object.
- ****kwargs** – keywords passed to the constructor.

Returns a new combined object.

Return type (*DomesticHotWaterSetting*)

extend(*other, allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other, attr, weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

validate()

Validate object and fill in missing values.

classmethod whole_building(idf)

Create one DomesticHotWaterSetting for whole building model.

Parameters **idf** (*IDF*) – The idf model.

Returns The DomesticHotWaterSetting object.

Return type *DomesticHotWaterSetting*

mapping(validate=True)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

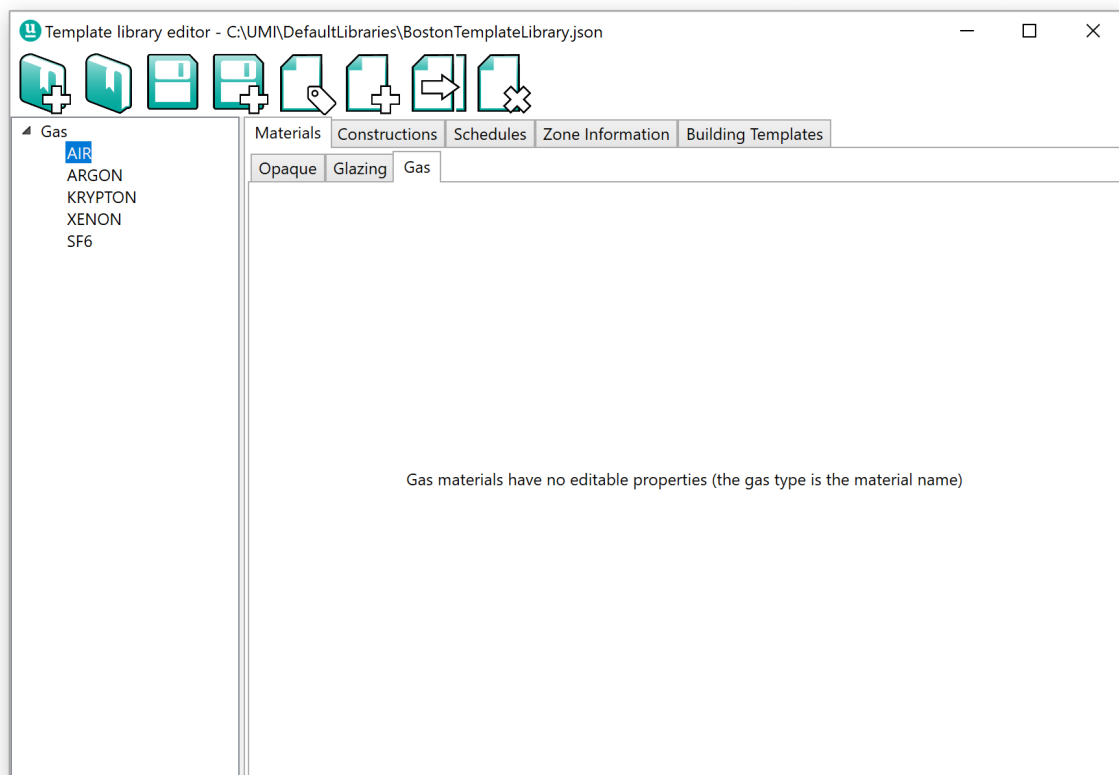
duplicate()

Get copy of self.

archetypal.template.GasMaterial

class archetypal.template.**GasMaterial**(*Name, Conductivity=None, Density=None, Category='Gases', **kwargs*)

Gas Materials.



Initialize object with parameters.

Parameters

- **Name** (*str*) – The name of the GasMaterial.
- **Conductivity** (*float*) – Thermal conductivity (W/m-K).
- **Density** (*float*) – A number representing the density of the material in kg/m3. This is essentially the mass of one cubic meter of the material.
- **Category** (*str*) – Category is set as “Gases” for GasMaterial.
- ****kwargs** – keywords passed to the MaterialBase constructor.

property Type

Get or set the gas type.

Choices are (“Air”, “Argon”, “Krypton”, “Xenon”).

property Conductivity

Get or set the conductivity of the gas at 0C [W/m-K].

property Density

Get or set the density of the gas.

property molecular_weight

Get the molecular weight [kg/mol].

property specific_heat

Get the material layer’s specific heat at 0C [J/kg-K].

duplicate()

Get copy of self.

classmethod from_dict(data, **kwargs)

Create a GasMaterial from a dictionary.

Parameters

- **data** (*dict*) – A python dictionary.
- ****kwargs** – keywords passed the MaterialBase constructor.

to_dict()

Return GasMaterial dictionary representation.

to_epbunch(idf, thickness)

Convert self to an epbunch given an idf model and a thickness.

Parameters

- **idf** (*IDF*) – An IDF model.
- **thickness** (*float*) – the thickness of the material.

```
WindowMaterial:Gas,
    AIR_0.006_B_Dbl_Air_Cl,    !- Name
    AIR,                        !- Gas Type
    0.006;                     !- Thickness
```

Returns The EpBunch object added to the idf model.

Return type EpBunch

mapping(validate=True)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

density_at_temperature(*t_kelvin*, *pressure=101325*)

Get the density of the gas [kg/m3] at a given temperature and pressure.

This method uses CoolProp to get the density.

Parameters

- **t_kelvin** (*float*) – The average temperature of the gas cavity in Kelvin.
- **pressure** (*float*) – The average pressure of the gas cavity in Pa. Default is 101325 Pa for standard pressure at sea level.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property Cost

Get or set the cost of the material [\$].

property DataSource

Get or set the datasource of the object.

property EmbodiedCarbon

Get or set the embodied carbon of the material [kgCO2/kg].

property EmbodiedEnergy

Get or set the embodied energy of the material [MJ/kg].

property Name

Get or set the name of the object.

property SubstitutionRatePattern

Get or set the substitution rate pattern of the material.

property SubstitutionTimestep

Get or set the substitution timestep of the material.

property TransportCarbon

Get or set the transportation carbon of the material [kgCO2/kg/km].

property TransportDistance

Get or set the transportation distance of the material [km].

property TransportEnergy

Get or set the transportation energy of the material [MJ/kg/km].

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

specific_heat_at_temperature(*t_kelvin, pressure=101325*)

Get the specific heat of the gas [J/(kg-K)] at a given Kelvin temperature.

This method uses CoolProp to get the density.

Parameters

- **t_kelvin** (*float*) – The average temperature of the gas cavity in Kelvin.
- **pressure** (*float*) – The average pressure of the gas cavity in Pa. Default is 101325 Pa for standard pressure at sea level.

to_ref()

Return a ref pointer to self.

validate()

Validate object and fill in missing values.

viscosity_at_temperature(*t_kelvin, pressure=101325*)

Get the viscosity of the gas [kg/m-s] at a given Kelvin temperature.

This method uses CoolProp to get the density.

Parameters

- **t_kelvin** (*float*) – The average temperature of the gas cavity in Kelvin.
- **pressure** (*float*) – The average pressure of the gas cavity in Pa. Default is 101325 Pa for standard pressure at sea level.

conductivity_at_temperature(*t_kelvin, pressure=101325*)

Get the conductivity of the gas [W/(m-K)] at a given Kelvin temperature.

This method uses CoolProp to get the density. Note that the thermal conductivity model is not available for Krypton, Xenon gases. Values from the literature are used instead.

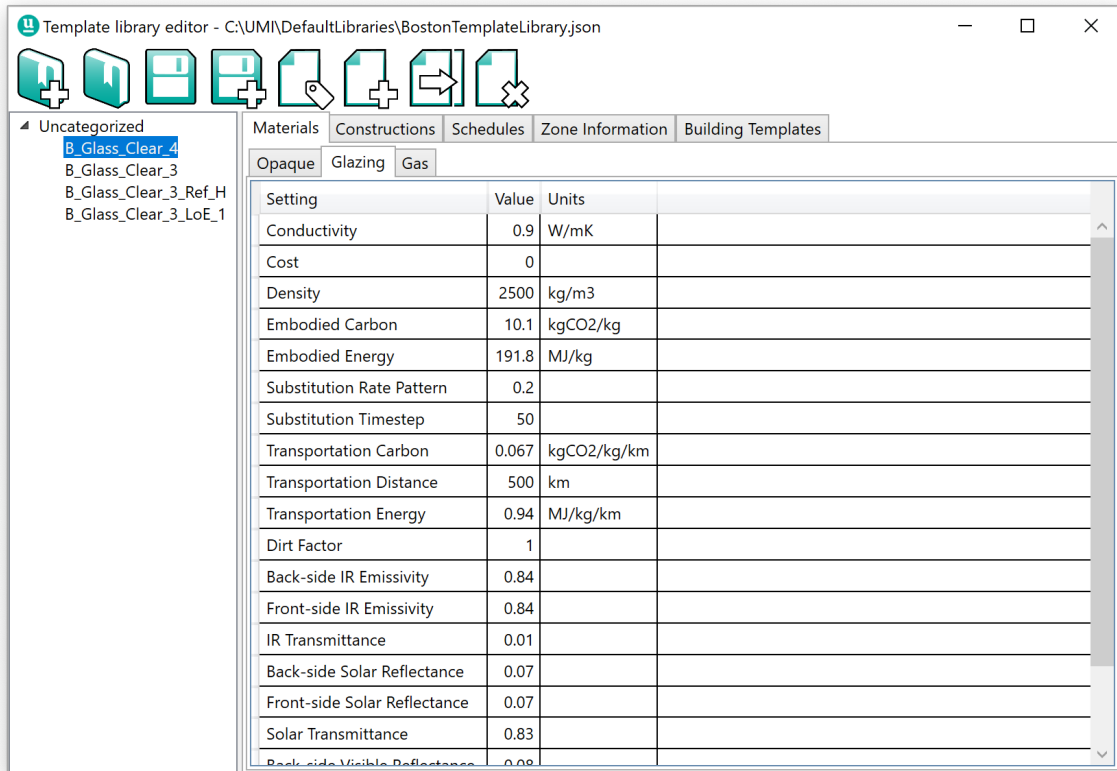
Parameters

- **t_kelvin** (*float*) – The average temperature of the gas cavity in Kelvin.
- **pressure** (*float*) – The average pressure of the gas cavity in Pa. Default is 101325 Pa for standard pressure at sea level.

archetypal.template.GlazingMaterial

```
class archetypal.template.GlazingMaterial(Name, Density=2500, Conductivity=0.9,
                                           SolarTransmittance=0, SolarReflectanceFront=0,
                                           SolarReflectanceBack=0, VisibleTransmittance=0,
                                           VisibleReflectanceFront=0, VisibleReflectanceBack=0,
                                           IRTTransmittance=0, IREmissivityFront=0,
                                           IREmissivityBack=0, DirtFactor=1.0, Cost=0.0, **kwargs)
```

Glazing Materials class.



Initialize a GlazingMaterial object.

Parameters

- **Name** (*str*) – The name of the GlazingMaterial.
- **Density** (*float*) – A number representing the density of the material in kg/m3. This is essentially the mass of one cubic meter of the material.
- **Conductivity** (*float*) – Thermal conductivity (W/m-K).
- **SolarTransmittance** (*float*) – Transmittance at normal incidence averaged over the solar spectrum.
- **SolarReflectanceFront** (*float*) – Front-side reflectance at normal incidence averaged over the solar spectrum.
- **SolarReflectanceBack** (*float*) – Back-side reflectance at normal incidence averaged over the solar spectrum.

- **VisibleTransmittance** (*float*) – Transmittance at normal incidence averaged over the solar spectrum and weighted by the response of the human eye.
- **VisibleReflectanceFront** (*float*) – Front-side reflectance at normal incidence averaged over the solar spectrum and weighted by the response of the human eye.
- **VisibleReflectanceBack** (*float*) – Back-side reflectance at normal incidence averaged over the solar spectrum and weighted by the response of the human eye.
- **IRTransmittance** (*float*) – Long-wave transmittance at normal incidence.
- **IREmissivityFront** (*float*) – Front-side long-wave emissivity.
- **IREmissivityBack** (*float*) – Back-side long-wave emissivity.
- **DirtFactor** (*float*) – This is a factor that corrects for the presence of dirt on the glass. Using a material with dirt correction factor < 1.0 in the construction for an interior window will result in an error message.
- ****kwargs** – keywords passed to the MaterialBase constructor. For more info, see MaterialBase.

property Conductivity

Get or set the conductivity of the material [W/m-K].

property Density

Get or set the density of the material [J/kg-K].

property DirtFactor

Get or set the dirt correction factor [-].

property IREmissivityBack

Get or set the infrared emissivity of the back side [-].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property Cost

Get or set the cost of the material [\$].

property DataSource

Get or set the datasource of the object.

property EmbodiedCarbon

Get or set the embodied carbon of the material [kgCO2/kg].

property EmbodiedEnergy

Get or set the embodied energy of the material [MJ/kg].

property IREmissivityFront

Get or set the infrared emissivity of the front side [-].

property Name

Get or set the name of the object.

property SubstitutionRatePattern

Get or set the substitution rate pattern of the material.

property SubstitutionTimestep

Get or set the substitution timestep of the material.

property TransportCarbon

Get or set the transportation carbon of the material [kgCO2/kg/km].

property TransportDistance

Get or set the transportation distance of the material [km].

property TransportEnergy

Get or set the transporation energy of the material [MJ/kg/km].

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* ([UmiBase](#)) –

Returns self

Return type [UmiBase](#)

float_mean(*other*, *attr*, *weights*=None)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** ([UmiBase](#)) – The other UmiBase object to calculate average value with.
- **attr** ([str](#)) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* ([str](#)) – the name.

to_ref()

Return a ref pointer to self.

validate()

Validate object and fill in missing values.

property IRTransmittance

Get or set the infrared transmittance [-].

property VisibleReflectanceBack

Get or set the visible reflectance of the back side [-].

property VisibleReflectanceFront

Get or set the visible reflectance of the front side [-].

property VisibleTransmittance

Get or set the visible transmittance [-].

property SolarReflectanceBack

Get or set the solar reflectance of the back side [-].

property SolarReflectanceFront

Get or set the solar reflectance of the front side [-].

property SolarTransmittance

Get or set the solar transmittance [-].

combine(*other*, *weights=None*, *allow_duplicates=False*)

Combine two GlazingMaterial objects together.

Parameters *other* (*GlazingMaterial*) – The other GlazingMaterial object to combine with.

Returns the combined GlazingMaterial object.

Return type (*GlazingMaterial*)

to_dict()

Return GlazingMaterial dictionary representation.

to_epbunch(*idf*, *thickness*) → *geomeppy.patches.EpBunch*

Convert self to an EpBunch given an idf model and a thickness.

Parameters

- **idf** (*IDF*) – An IDF model.
- **thickness** (*float*) – the thickness of the material.

```
WindowMaterial:Glazing,
  B_Glass_Clear_3_0.003_B_Dbl_Air_Cl,    !- Name
  SpectralAverage,                        !- Optical Data Type
  SpectralAverage,                        !- Window Glass Spectral Data Set Name
  0.003,                                  !- Thickness
  0.83,                                  !- Solar Transmittance at Normal Incidence
  0.07,                                  !- Front Side Solar Reflectance at Normal
↪Incidence
  0.07,                                  !- Back Side Solar Reflectance at Normal Incidence
  0.89,                                  !- Visible Transmittance at Normal Incidence
  0.08,                                  !- Front Side Visible Reflectance at Normal
↪Incidence
  0.08,                                  !- Back Side Visible Reflectance at Normal
↪Incidence
  0,                                      !- Infrared Transmittance at Normal Incidence
  0.84,                                  !- Front Side Infrared Hemispherical Emissivity
  0.84,                                  !- Back Side Infrared Hemispherical Emissivity
  0.9,                                    !- Conductivity
  1;                                      !- Dirt Correction Factor for Solar and Visible
↪Transmittance
```

Returns The EpBunch object added to the idf model.

Return type *EpBunch*

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

classmethod from_dict(*data*, ***kwargs*)

Create a GlazingMaterial from a dictionary.

Parameters **data** – The python dictionary following the format below.

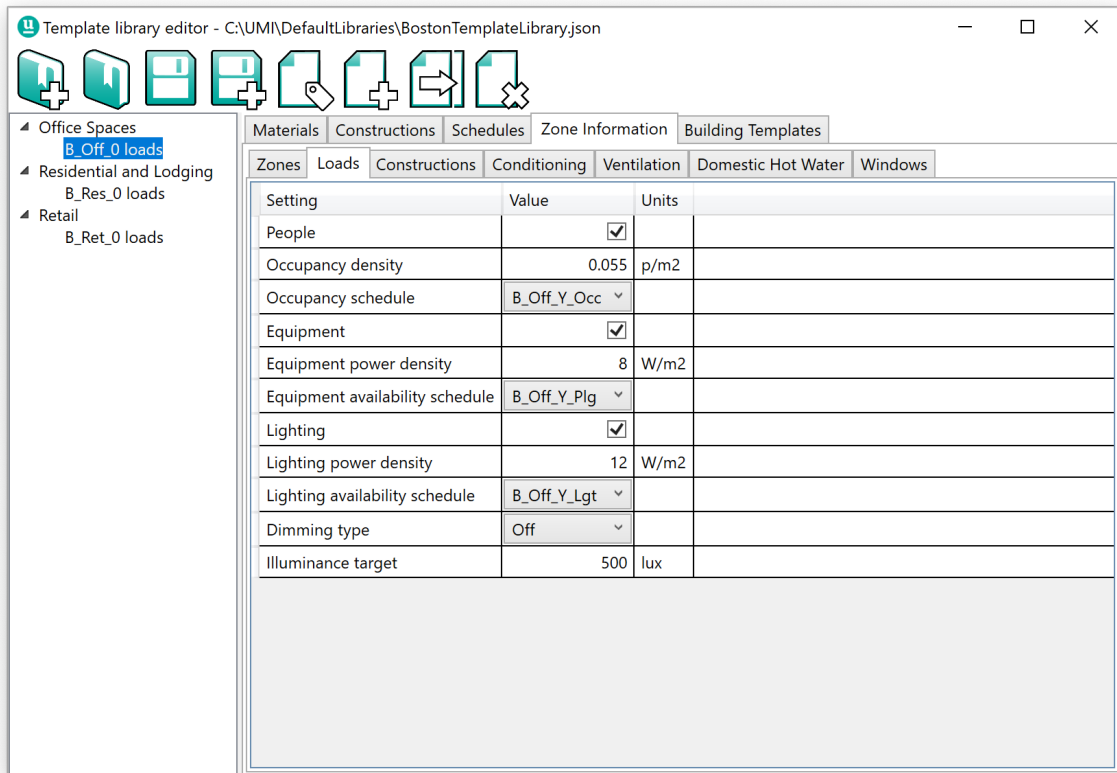
```
{'$id': '1',
 'DirtFactor': 1.0,
 'IREmissivityBack': 0.0,
 'IREmissivityFront': 0.0,
 'IRTransmittance': 0.0,
 'SolarReflectanceBack': 0.0,
 'SolarReflectanceFront': 0.0,
 'SolarTransmittance': 0.0,
 'VisibleReflectanceBack': 0.0,
 'VisibleReflectanceFront': 0.0,
 'VisibleTransmittance': 0.0,
 'Conductivity': 0.0,
 'Cost': 0.0,
 'Density': 2500,
 'EmbodiedCarbon': 0.0,
 'EmbodiedEnergy': 0.0,
 'SubstitutionRatePattern': [1.0],
 'SubstitutionTimestep': 100.0,
 'TransportCarbon': 0.0,
 'TransportDistance': 0.0,
 'TransportEnergy': 0.0,
 'Category': 'Uncategorized',
 'Comments': '',
 'DataSource': None,
 'Name': 'A'}
```

archetypal.template.ZoneLoad

class archetypal.template.**ZoneLoad**(*Name*, *EquipmentPowerDensity*=0,
 EquipmentAvailabilitySchedule=None, *LightingPowerDensity*=0,
 LightsAvailabilitySchedule=None, *PeopleDensity*=0,
 OccupancySchedule=None, *IsEquipmentOn*=True,
 IsLightingOn=True, *IsPeopleOn*=True,
 DimmingType=DimmingTypes.Continuous, *IlluminanceTarget*=500,
 area=1, *volume*=1, ***kwargs*)

Zone Loads.

Important: Please note that the calculation of the equipment power density will sum up the electric equipment objects as well as the gas equipment objects.



Initialize a new ZoneLoad object.

Parameters

- **DimmingType** (*int*) – Different types to dim the lighting to respect the IlluminanceTarget and taking into account the daylight illuminance:
 - Continuous = 0, the overhead lights dim continuously and linearly from (maximum electric power, maximum light output) to (minimum electric power, minimum light output) as the daylight illuminance increases. The lights stay on at the minimum point with further increase in the daylight illuminance.
 - Off = 1, Lights switch off completely when the minimum dimming point is reached.
 - Stepped = 2, the electric power input and light output vary in discrete, equally spaced steps.
- **EquipmentAvailabilitySchedule** (*UmiSchedule*) – The name of the schedule (Day | Week | Year) that modifies the design level parameter for electric equipment.
- **EquipmentPowerDensity** (*float*) – Equipment Power Density in the zone (W/m²).
- **IlluminanceTarget** (*float*) – Number of lux to be respected in the zone
- **LightingPowerDensity** (*float*) – Lighting Power Density in the zone (W/m²).
- **LightsAvailabilitySchedule** (*UmiSchedule*) – The name of the schedule (Day | Week | Year) that modifies the design level parameter for lighting.
- **OccupancySchedule** (*UmiSchedule*) – The name of the schedule (Day | Week | Year) that modifies the number of people parameter for electric equipment.

- **IsEquipmentOn** (*bool*) – If True, heat gains from Equipment are taken into account for the zone’s load calculation.
- **IsLightingOn** (*bool*) – If True, heat gains from Lights are taken into account for the zone’s load calculation.
- **IsPeopleOn** (*bool*) – If True, heat gains from People are taken into account for the zone’s load calculation.
- **PeopleDensity** (*float*) – Density of people in the zone (people/m²).
- **area** (*float*) – The floor area associated to this zone load object.
- ****kwargs** – Other keywords passed to the parent constructor UmiBase.

property DimmingType

Get or set the dimming type.

Hint: To set the value an int or a string is supported. Choices are (<DimmingTypes.Continuous: 0>, <DimmingTypes.Off: 1>, <DimmingTypes.Stepped: 2>)

property EquipmentAvailabilitySchedule

Get or set the equipment availability schedule.

property EquipmentPowerDensity

Get or set the equipment power density [W/m²].

property IlluminanceTarget

Get or set the illuminance target [lux].

property LightingPowerDensity

Get or set the lighting power density [W/m²].

property LightsAvailabilitySchedule: `archetypal.template.schedule.UmiSchedule`

Get or set the lights availability schedule.

property OccupancySchedule: `archetypal.template.schedule.UmiSchedule`

Get or set the occupancy schedule.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

property PeopleDensity

Get or set the people density [ppl/m²].

property IsEquipmentOn

Get or set the use of equipment [bool].

property IsLightingOn

Get or set the use of lighting [bool].

property IsPeopleOn

Get or set people [bool].

property area

Get or set the floor area of the zone associated to this zone load [m²].

property volume

Get or set the volume of the zone associated to this zone load [m³].

classmethod from_dict(*data*, *schedules*, ***kwargs*)

Create a ZoneLoad from a dictionary.

Parameters

- **data** (*dict*) – A python dictionary with the structure shown bellow.
- **schedules** (*dict*) – A python dictionary of UmiSchedules with their id as keys.
- ****kwargs** – keywords passed to parent constructors.

```
{
  "$id": "172",
  "DimmingType": 1,
  "EquipmentAvailabilitySchedule": {
```

(continues on next page)

(continued from previous page)

```

    "$ref": "147"
  },
  "EquipmentPowerDensity": 8.0,
  "IlluminanceTarget": 500.0,
  "LightingPowerDensity": 12.0,
  "LightsAvailabilitySchedule": {
    "$ref": "146"
  },
  "OccupancySchedule": {
    "$ref": "145"
  },
  "IsEquipmentOn": true,
  "IsLightingOn": true,
  "IsPeopleOn": true,
  "PeopleDensity": 0.055,
  "Category": "Office Spaces",
  "Comments": null,
  "DataSource": "MIT_SDL",
  "Name": "B_Off_0 loads"
},

```

combine(*other*, *weights=None*)

Combine two ZoneLoad objects together. Returns a new object.

Parameters

- **other** (*ZoneLoad*) – The other ZoneLoad object.
- **weights** (*list-like*, *optional*) – A list-like object of len 2. If None, the *settings.zone_weight* of the objects is used.

Returns the combined ZoneLoad object.

Return type (*ZoneLoad*)

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

to_dict()

Return ZoneLoad dictionary representation.

to_epbunch(*idf*, *zone_name*)

Convert the zone load to epbunch given an idf model and a zone name.

Parameters

- **idf** (*IDF*) – The idf model. epbunches will be added to this model.
- **zone_name** (*str*) – The name of the zone in the idf model.

```

People,
  People Perim,           !- Name
  Perim,                  !- Zone or ZoneList Name
  B_Off_Y_Occ,            !- Number of People Schedule Name

```

(continues on next page)

(continued from previous page)

```

    People/Area,          !- Number of People Calculation Method
    ,                    !- Number of People
    0.055,                !- People per Zone Floor Area
    ,                    !- Zone Floor Area per Person
    0.3,                  !- Fraction Radiant
    AUTOCALCULATE,        !- Sensible Heat Fraction
    PerimPeopleActivity,   !- Activity Level Schedule Name
    3.82e-08,             !- Carbon Dioxide Generation Rate
    No,                   !- Enable ASHRAE 55 Comfort Warnings
    ZoneAveraged,          !- Mean Radiant Temperature Calculation Type
    ,                    !- Surface NameAngle Factor List Name
    PerimWorkEfficiency,   !- Work Efficiency Schedule Name
    DynamicClothingModelASHRAE55, !- Clothing Insulation Calculation Method
    ,                    !- Clothing Insulation Calculation Method
    ↪Schedule Name
    ,                    !- Clothing Insulation Schedule Name
    PerimAirVelocity,      !- Air Velocity Schedule Name
    AdaptiveASH55;         !- Thermal Comfort Model 1 Type

Lights,
    Perim General lighting, !- Name
    Perim,                 !- Zone or ZoneList Name
    B_Off_Y_Lgt,           !- Schedule Name
    Watts/Area,            !- Design Level Calculation Method
    ,                      !- Lighting Level
    12,                    !- Watts per Zone Floor Area
    ,                      !- Watts per Person
    0,                     !- Return Air Fraction
    0.42,                  !- Fraction Radiant
    0.18,                  !- Fraction Visible
    1,                     !- Fraction Replaceable
    ;                      !- EndUse Subcategory

ElectricEquipment,
    Perim Equipment 1,     !- Name
    Perim,                 !- Zone or ZoneList Name
    B_Off_Y_Plg,           !- Schedule Name
    Watts/Area,            !- Design Level Calculation Method
    ,                      !- Design Level
    8,                     !- Watts per Zone Floor Area
    ,                      !- Watts per Person
    0,                     !- Fraction Latent
    0.2,                   !- Fraction Radiant
    0,                     !- Fraction Lost
    ;                      !- EndUse Subcategory

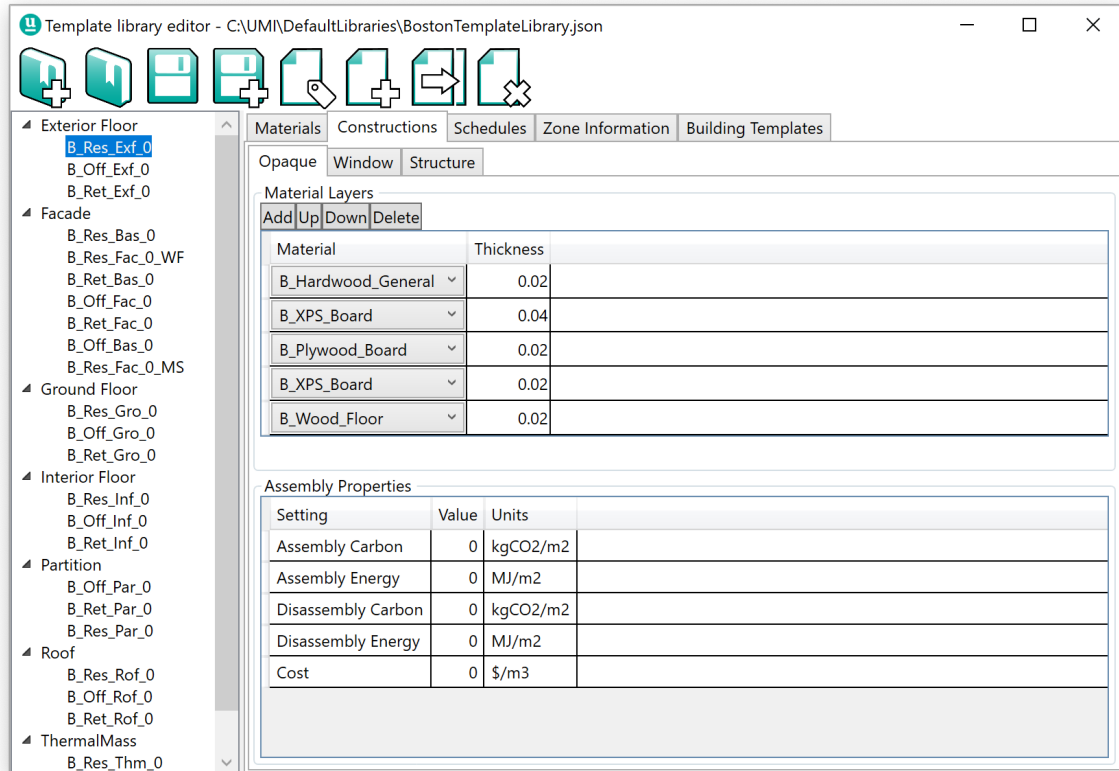
```

Returns The EpBunch object added to the idf model.

Return type EpBunch

archetypal.template.OpaqueConstruction

class archetypal.template.OpaqueConstruction(*Name*, *Layers*, ****kwargs**)
 Opaque Constructions.



Properties:

- r_value
- u_value
- r_factor
- u_factor
- equivalent_heat_capacity_per_unit_volume
- specific_heat
- heat_capacity_per_unit_wall_area
- total_thickness
- mass_per_unit_area
- timeconstant_per_unit_area
- solar_reflectance_index

Initialize an OpaqueConstruction.

Parameters

- **Layers** (*list of archetypal.MaterialLayer*) – List of MaterialLayers making up the construction.
- ****kwargs** – Other attributes passed to parent constructors such as ConstructionBase.

property r_value

Get or set the thermal resistance [Km²/W] (excluding air films).

Note that, when setting the R-value, the thickness of the inferred insulation layer will be adjusted.

property equivalent_heat_capacity_per_unit_volume

Get the equivalent per unit wall volume heat capacity [J/(kgK)].

Hint: “The physical quantity which represents the heat storage capability is the wall heat capacity, defined as $HC=M \cdot c$. While the per unit wall area of this quantity is $(HC/A)=\bar{c}$, where $\bar{c}$ is the wall thickness, the per unit volume wall heat capacity, being a characteristic wall quantity independent from the wall thickness, is equal to $\cdot c$. This quantity for a composite wall of an overall thickness L , is usually defined as the equivalent per unit wall volume heat capacity and it is expressed as $(c)_{eq} = (1/L) \sum_{i=1}^n (c_i \cdot l_i)$ where l_i , c_i and ρ_i are the densities, the specific heat capacities and the layer thicknesses of the n parallel layers of the composite wall.” [ref]

property specific_heat

Get the construction specific heat weighted by wall area mass [J/(kgK)].

property heat_capacity_per_unit_wall_area

Get the construction heat capacity per unit wall area [J/(m²K)].

Hint: $(HC/A) = \bar{c}$, where $\bar{c}$ is the wall thickness.

property total_thickness

Get the construction total thickness [m].

property mass_per_unit_area

Get the construction mass per unit area [kg/m²].

property time_constant_per_unit_area

Get the construction time constant per unit area [seconds/m²].

property solar_reflectance_index

Get the Solar Reflectance Index of the exposed surface.

Hint: calculation from K-12 AEDG, derived from ASTM E1980 assuming medium wind speed.

infer_insulation_layer()

Return the material layer index that corresponds to the insulation layer.

combine(other, method='dominant_wall', allow_duplicates=False)

Combine two OpaqueConstruction together.

Parameters

- **other** (*OpaqueConstruction*) – The other OpaqueConstruction object to combine with.
- **method** (*str*) – Equivalent wall assembly method. Only ‘dominant_wall’ is safe to use. ‘constant_ufactor’ is still weird in terms of respecting the thermal response of the walls and may cause conversion issues with Conduction Transfer Functions (CTFs) in EnergyPlus.

Returns the combined ZoneLoad object.

Return type (*OpaqueConstruction*)

dominant_wall(*other*, *weights*)

Return dominant wall construction between self and other.

Parameters

- **other** –
- **weights** –

classmethod from_dict(*data*, *materials*, ***kwargs*)

Create an OpaqueConstruction from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **materials** (*dict*) – A dictionary of materials with their id as keys.

```
materials = {} # dict of materials.
data = {
    "$id": "140300770659680",
    "Layers": [
        {
            "Material": {
                "$ref": "140300653743792"
            },
            "Thickness": 0.013
        },
        {
            "Material": {
                "$ref": "140300653743792"
            },
            "Thickness": 0.013
        }
    ],
    "AssemblyCarbon": 0.0,
    "AssemblyCost": 0.0,
    "AssemblyEnergy": 0.0,
    "DisassemblyCarbon": 0.0,
    "DisassemblyEnergy": 0.0,
    "Category": "Partition",
    "Comments": "",
    "DataSource": "ASHRAE 90.1-2007",
    "Name": "90.1-2007 Nonres 6A Int Wall"
}
```

classmethod generic_internalmass(***kwargs*)

Create a generic internal mass object.

Parameters ***kwargs* – keywords passed to the class constructor.

classmethod from_epbunch(*epbunch*, ***kwargs*)

Create an OpaqueConstruction object from an epbunch.

Possible keys are “BuildingSurface:Detailed” or “InternalMass”

Parameters

- **epbunch** (*EpBunch*) – The epbunch object.
- ****kwargs** – keywords passed to the LayeredConstruction constructor.

to_dict()

Return OpaqueConstruction dictionary representation.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

classmethod generic(***kwargs*)

Return OpaqueConstruction based on 90.1-2007 Nonres 4B Int Wall.

property AssemblyCarbon

Get or set the assembly carbon [kgCO2/m2].

property AssemblyCost

Get or set the assembly cost [\$ /m2].

property AssemblyEnergy

Get or set the assembly energy [MJ/m2].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property DisassemblyCarbon

Get or set the disassembly carbon [kgCO2/m2].

property DisassemblyEnergy

Get or set the disassembly energy [MJ/m2].

property Layers:

`List[Union[archetypal.template.materials.material_layer.MaterialLayer, archetypal.template.materials.gas_layer.GasLayer]]`

Get or set the material layers.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other, allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other, attr, weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

in_h(*t_kelvin=293.15, delta_t=15, height=1.0, angle=90, pressure=101325*)

Get the detailed indoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

in_h_c(*t_kelvin=293.15, delta_t=15, height=1.0, angle=90, pressure=101325*)

Get detailed indoor convective heat transfer coef. according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

in_h_simple()

Get the simple indoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

property inside_emissivity

Get the emissivity of the inside face of the construction [-].

out_h(*wind_speed*=6.7, *t_kelvin*=273.15)

Get the detailed outdoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **wind_speed** (*float*) – The average outdoor wind speed [m/s]. This affects the convective heat transfer coefficient. Default is 6.7 m/s.
- **t_kelvin** (*float*) – The average between the outdoor temperature and the exterior surface temperature. This can affect the radiative heat transfer. Default is 273.15K (0C).

out_h_simple()

Get the simple outdoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

property outside_emissivity

Get the hemispherical emissivity of the outside face of the construction.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

property r_factor

Get the R-factor [m2-K/W] (including air films).

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_epbunch(*idf*)

Get a Construction EpBunch given an idf model.

Notes

Will create layered materials as well.

Parameters **idf** (*IDF*) – An idf model to add the EpBunch in.

Returns The EpBunch object added to the idf model.

Return type EpBunch

to_ref()

Return a ref pointer to self.

property u_factor

Get the overall heat transfer coefficient (including air films) W/(m2K).

property u_value

Get the heat transfer coefficient [W/m2K] (excluding air films).

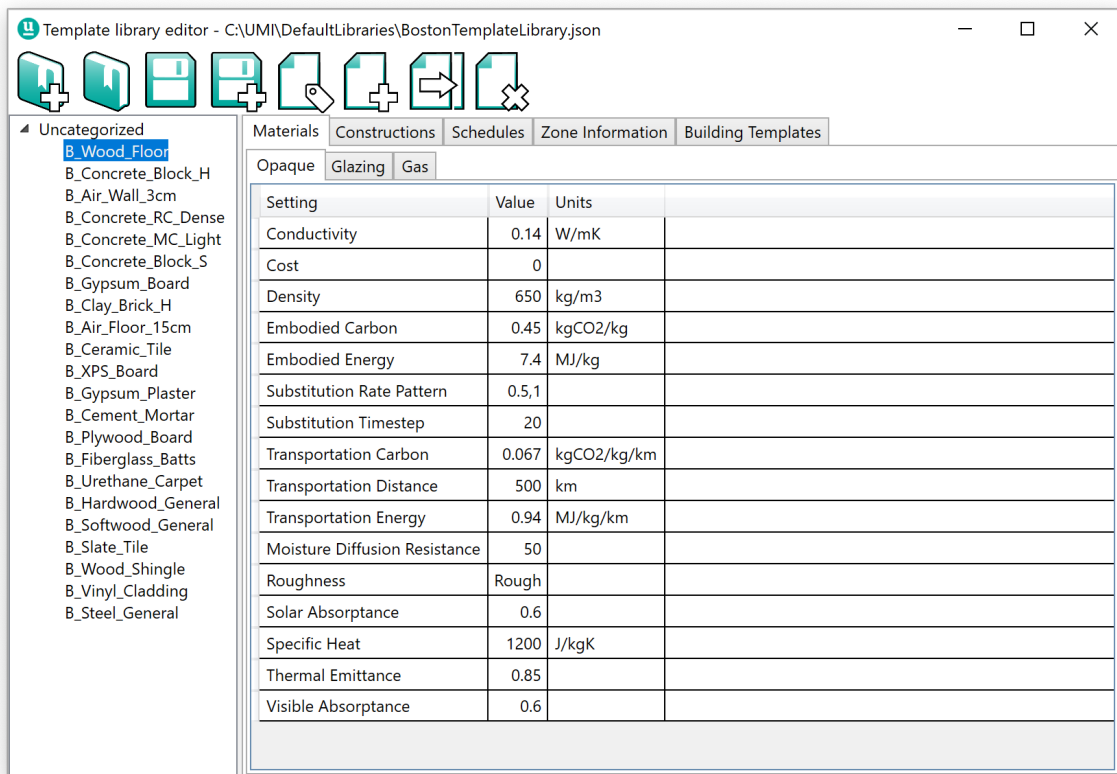
validate()

Validate object and fill in missing values.

archetypal.template.OpaqueMaterial

```
class archetypal.template.OpaqueMaterial(Name, Conductivity, SpecificHeat, SolarAbsorptance=0.7,
                                          ThermalEmittance=0.9, VisibleAbsorptance=0.7,
                                          Roughness='Rough', Cost=0, Density=1,
                                          MoistureDiffusionResistance=50, EmbodiedCarbon=0.45,
                                          EmbodiedEnergy=0, TransportCarbon=0,
                                          TransportDistance=0, TransportEnergy=0,
                                          SubstitutionRatePattern=None, SubstitutionTimestep=20,
                                          **kwargs)
```

Use this component to create a custom opaque material.



Initialize an opaque material.

Parameters

- **Name** (*str*) – The name of the material.
- **Conductivity** (*float*) – A number representing the conductivity of the material in W/m-K. This is essentially the heat flow in Watts across one meter thick of the material when the temperature difference on either side is 1 Kelvin. Modeling layers with conductivity higher than 5.0 W/(m-K) is not recommended.
- **SpecificHeat** (*float*) – A number representing the specific heat capacity of the material in J/kg-K. This is essentially the number of joules needed to raise one kg of the material by 1 degree Kelvin. Only values of specific heat of 100 or larger are allowed. Typical ranges are from 800 to 2000 J/(kg-K).
- **SolarAbsorptance** (*float*) – An number between 0 and 1 that represents the absorptance of solar radiation by the material. The default is set to 0.7, which is common for most non-

metallic materials.

- **ThermalEmittance** (*float*) – An number between 0 and 1 that represents the thermal absorptance of the material. The default is set to 0.9, which is common for most non-metallic materials. For long wavelength radiant exchange, thermal emissivity and thermal emittance are equal to thermal absorptance.
- **VisibleAbsorptance** (*float*) – An number between 0 and 1 that represents the absorptance of visible light by the material. The default is set to 0.7, which is common for most non-metallic materials.
- **Roughness** (*str*) – A text value that indicated the roughness of your material. This can be either “VeryRough”, “Rough”, “MediumRough”, “MediumSmooth”, “Smooth”, and “VerySmooth”. The default is set to “Rough”.
- **Density** (*float*) – A number representing the density of the material in kg/m3. This is essentially the mass of one cubic meter of the material.
- **MoistureDiffusionResistance** (*float*) – the factor by which the vapor diffusion in the material is impeded, as compared to diffusion in stagnant air [%].
- ****kwargs** – keywords passed to parent constructors.

property Conductivity

Get or set the conductivity of the material [W/m-K].

property Density

Get or set the density of the material [J/kg-K].

property Roughness

Get or set the roughness of the material.

Hint: choices are: “VeryRough”, “Rough”, “MediumRough”, “MediumSmooth”, “Smooth”, “VerySmooth”.

property SolarAbsorptance

Get or set the solar absorptance of the material [-].

property SpecificHeat

Get or set the specific heat of the material [J/(kg-K)].

property ThermalEmittance

Get or set the thermal emittance of the material [-].

property VisibleAbsorptance

Get or set the visible absorptance of the material [-].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property Cost

Get or set the cost of the material [\$].

property DataSource

Get or set the datasource of the object.

property EmbodiedCarbon

Get or set the embodied carbon of the material [kgCO2/kg].

property EmbodiedEnergy

Get or set the embodied energy of the material [MJ/kg].

property MoistureDiffusionResistance

Get or set the vapor resistance factor of the material [%].

property Name

Get or set the name of the object.

property SubstitutionRatePattern

Get or set the substitution rate pattern of the material.

property SubstitutionTimestep

Get or set the substitution timestep of the material.

property TransportCarbon

Get or set the transportation carbon of the material [kgCO2/kg/km].

property TransportDistance

Get or set the transportation distance of the material [km].

property TransportEnergy

Get or set the transporation energy of the material [MJ/kg/km].

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

classmethod generic(*kwargs*)**

Return a generic material based on properties of plaster board.

Parameters ****kwargs** – keywords passed to UmiBase constructor.

combine(*other*, *weights=None*, *allow_duplicates=False*)

Combine two OpaqueMaterial objects.

Parameters

- **weights** (*list-like*, *optional*) – A list-like object of len 2. If None, the density of the OpaqueMaterial of each objects is used as a weighting factor.
- **other** (*OpaqueMaterial*) – The other OpaqueMaterial object the combine with.

Returns A new combined object made of self + other.

Return type *OpaqueMaterial*

to_ref()

Return a ref pointer to self.

to_dict()

Return OpaqueMaterial dictionary representation.

classmethod from_dict(*data*, ****kwargs**)

Create an OpaqueMaterial from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- ****kwargs** – keywords passed to MaterialBase constructor.

```
{
    "$id": "1",
    "MoistureDiffusionResistance": 50.0,
    "Roughness": "Rough",
    "SolarAbsorptance": 0.7,
    "SpecificHeat": 920.0,
    "ThermalEmittance": 0.9,
    "VisibleAbsorptance": 0.7,
    "Conductivity": 0.85,
    "Cost": 0.0,
    "Density": 2000,
    "EmbodiedCarbon": 0.45,
    "EmbodiedEnergy": 0.0,
    "SubstitutionRatePattern": [
        1.0
    ],
    "SubstitutionTimestep": 20.0,
    "TransportCarbon": 0.0,
    "TransportDistance": 0.0,
    "TransportEnergy": 0.0,
    "Category": "Uncategorized",
    "Comments": "",
    "DataSource": null,
    "Name": "Concrete"
}
```

classmethod from_epbunch(*epbunch*, ****kwargs**)

Create an OpaqueMaterial from an EpBunch.

Note that “Material”, “Material:NoMAss” and “Material:AirGap” objects are supported.

Hint: (From EnergyPlus Manual): When a user enters such a “no mass” material into EnergyPlus, internally the properties of this layer are converted to approximate the properties of air (density, specific heat, and conductivity) with the thickness adjusted to maintain the user’s desired R-Value. This allowed such layers to be handled internally in the same way as other layers without any additional changes to the code. This solution was deemed accurate enough as air has very little thermal mass and it made the coding of the state space method simpler.

For Material:AirGap, a similar strategy is used, with the exception that solar properties (solar and visible absorptance and emittance) are assumed null.

Parameters

- **epbunch** (*EpBunch*) – EP-Construction object
- ****kwargs** –

to_epbunch(*idf*, *thickness*) → *eppy.bunch_subclass.EpBunch*

Convert self to an EpBunch given an idf model and a thickness.

Parameters

- **idf** (*IDF*) – An IDF model.
- **thickness** (*float*) – the thickness of the material.

```
MATERIAL,
,           !- Name
,           !- Roughness
,           !- Thickness
,           !- Conductivity
,           !- Density
,           !- Specific Heat
0.9,       !- Thermal Absorptance
0.7,       !- Solar Absorptance
0.7;
```

Returns The EpBunch object added to the idf model.

Return type EpBunch

validate()

Validate object and fill in missing values.

Hint: Some OpaqueMaterial don’t have a default value, therefore an empty string is parsed. This breaks the UmiTemplate Editor, therefore we set a value on these attributes (if necessary) in this validation step.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

duplicate()

Get copy of self.

archetypal.template.UmiSchedule

class archetypal.template.**UmiSchedule**(*Name*, *quantity=None*, ***kwargs*)

Class that handles Schedules.

Initialize object with parameters.

Parameters

- **Name** –
- **quantity** –
- ****kwargs** –

property quantity

Get or set the schedule quantity.

classmethod **constant_schedule**(*value=1*, *Name='AlwaysOn'*, *Type='Fraction'*, ***kwargs*)

Create an UmiSchedule with a constant value at each timestep.

Parameters

- **Type** –
- **value** (*float*) –
- **Name** –
- **idf** –
- ****kwargs** –

classmethod **random**(*Name='AlwaysOn'*, *Type='Fraction'*, ***kwargs*)

Create an UmiSchedule with a randomized value (0-1) at each timestep.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Type** (*str* or *ScheduleTypeLimits*) –
- ****kwargs** – keywords passed to the constructor.

classmethod **from_values**(*Name*, *Values*, *Type='Fraction'*, ***kwargs*)

Create an UmiSchedule from a list of values.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Values** (*list*) –
- **Type** –
- ****kwargs** –

combine(*other*, *weights=None*, *quantity=None*)

Combine two UmiSchedule objects together.

Parameters

- **other** (*UmiSchedule*) – The other Schedule object to combine with.
- **weights** (*list*, *dict* or *string*) – Attribute of self and other containing the weight factor. If a list is passed, it must have len = 2; the first element is applied to self and the second element is applied to other. If a dict is passed, the self.Name and other.Name are the keys. If a str is passed, the

- **quantity**(*list or dict or bool*) – Scalar value that will be multiplied by self before the averaging occurs. This ensures that the resulting schedule returns the correct integrated value. If a dict is passed, keys are schedules Names and values are quantities.

Returns the combined UmiSchedule object.

Return type (*UmiSchedule*)

Raises **TypeError** – if Quantity is not of type list, tuple, dict or a callable.

develop()

Develop the UmiSchedule into a Year-Week-Day schedule structure.

get_unique()

Return the first of all the created objects that is equivalent to self.

to_dict()

Return UmiSchedule dictionary representation.

Hint: UmiSchedule does not implement the to_dict method because it is not used when generating the json file. Only Year-Week- and DaySchedule classes are used.

to_ref()

Return a ref pointer to self.

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

get_ref(*ref*)

Get item matching reference id.

Parameters **ref** –

duplicate()

Get copy of self.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the schedule.

property Type

Get or set the schedule type limits object. Can be None.

property Values

Get or set the list of schedule values.

property all_values: **numpy.ndarray**

Return numpy array of schedule Values.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

classmethod from_epbunch(*epbunch*, *strict=False*, *Type=None*, ***kwargs*)

Create a Schedule from an epbunch.

Parameters

- **epbunch** –
- **strict** –
- ****kwargs** –

static get_schedule_type_limits_name(*epbunch*)

Return the Schedule Type Limits name associated to this schedule.

property id

Get or set the id.

property max

Get the maximum value of the schedule.

property mean

Get the mean value of the schedule.

property min

Get the minimum value of the schedule.

plot(***kwargs*)

Plot the schedule. Implements the .loc accessor on the series object.

Notes

Plotting can also be achieved through the series property: *Schedule.series.plot()*.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF()
>>> epbunch = idf.schedules_dict["NECB-A-Thermostat Setpoint-Heating"]
>>> s = Schedule.from_epbunch(epbunch)
>>> )
>>> s.plot(drawstyle="steps-post")
```

Parameters ****kwargs** (*dict*) – keyword arguments passed to `EnergySeries.plot()`.

`plot2d(**kwargs)`

Plot the carpet plot of the schedule. Plot rectangular data as a color-encoded 2d-matrix.

Parameters

- **periodlength** (*int*) – The period length to show on the yaxis. By default, the period length will be calculated to represent one day of data on the yaxis and days on the xaxis.
- **vmin** (*float*) – The data value that defines 0.0 in the normalization. Defaults to the min value of the dataset.
- **vmax** (*float*) – The data value that defines 1.0 in the normalization. Defaults to the the max value of the dataset.
- **vcenter** (*float*) – The data value that defines 0.5 in the normalization.
- **axis_off** (*bool*) – If True, no axis is plotted.
- **cmap** (*str* or *Colormap*) – Colormap to select colors from. If string, load colormap with that name from matplotlib.
- **figsize** (*tuple*) – Size of a figure object. A tuple (width, height) in inches.
- **show** (*bool*) – whether to display the figure or not.
- **save** (*bool*) – whether to save the figure to disk or not.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg').
- **colorbar** (*bool*) – If True, plot the colorbar.
- **ax** (*Axes*) – An axes of the current figure.
- **filename** (*string*) – the name of the file to save.
- **extent** (*str* or *Bbox*) – Bounding box in inches: only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- **ylabel** (*str*) – Set the label for the y-axis.
- **xlabel** (*str*) – Set the label for the x-axis.
- **ax_title** (*bool* or *str*) – Title to use for the ax. If True, the Series name is used.
- ****kwargs** – Options to pass to matplotlib imshow method.

Returns

tuple containing:

fig: is the Matplotlib Figure object ax: a single axis object.

Return type (tuple)

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(name)

renames self as well as the cached object

Parameters name (*str*) – the name.

replace(new_values: *pandas.core.series.Series*)

Replace values with new values while keeping the full load hours constant.

Time steps that are not specified in *new_values* will be adjusted to keep the full load hours of the schedule constant. No check whether the new schedule stays between the bounds set by self.Type is done. Be aware.

scale(diversity=0.1)

Scale the schedule values by a diversity factor around the average.

property series

Return an EnergySeries.

property startDate

Get the start date of the schedule. Satisfies *startDayOfTheWeek*.

to_year_week_day()

Convert to three-tuple epbunch given an idf model.

Returns ‘Schedule:Year’, ‘Schedule:Week:Daily’ and ‘Schedule:Day:Hourly’ representations.

Returns

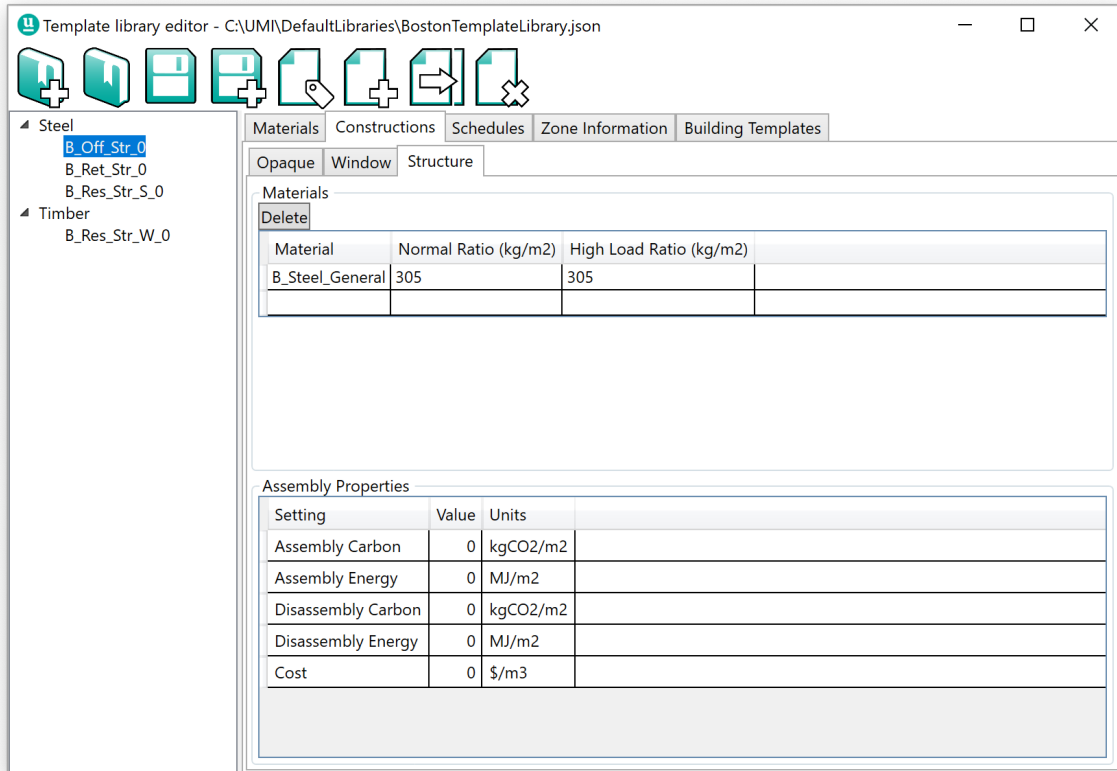
3-element tuple containing

- **yearly** (*Schedule*): The yearly schedule object
- **weekly** (*list of Schedule*): The list of weekly schedule objects
- **daily** (*list of Schedule*):The list of daily schedule objects

archetypal.template.StructureInformation

class archetypal.template.**StructureInformation**(Name, MassRatios, **kwargs)

Building Structure settings.



Initialize object.

Parameters

- **MassRatios** (*list of MassRatio*) – MassRatio object.
- ****kwargs** – keywords passed to the ConstructionBase constructor.

property MassRatios

Get or set the list of MassRatios.

property AssemblyCarbon

Get or set the assembly carbon [kgCO2/m2].

property AssemblyCost

Get or set the assembly cost [\$/m2].

property AssemblyEnergy

Get or set the assembly energy [MJ/m2].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property DisassemblyCarbon

Get or set the disassembly carbon [kgCO2/m2].

property DisassemblyEnergy

Get or set the disassembly energy [MJ/m2].

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

classmethod from_dict(*data*, *materials*, *kwargs*)**

Create StructureInformation from a dictionary.

Parameters

- **data** (*dict*) – A python dictionary.
- **materials** (*dict*) – A dictionary of python OpaqueMaterials with their id as keys.
- ****kwargs** – keywords passed to parent constructors.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

to_ref()

Return a ref pointer to self.

to_dict()

Return StructureInformation dictionary representation.

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

duplicate()

Get copy of self.

archetypal.template.VentilationSetting

```
class archetypal.template.VentilationSetting(Name, Infiltration=0.1, IsInfiltrationOn=True,
                                             IsNatVentOn=False, NatVentSchedule=None,
                                             IsWindOn=False, IsBuoyancyOn=True,
                                             NatVentMaxOutdoorAirTemp=30,
                                             NatVentMaxRelHumidity=90,
                                             NatVentMinOutdoorAirTemp=0,
                                             NatVentZoneTempSetpoint=18,
                                             ScheduledVentilationAch=0.6,
                                             ScheduledVentilationSchedule=None,
                                             ScheduledVentilationSetpoint=18,
                                             IsScheduledVentilationOn=False,
                                             VentilationType=VentilationType.Exhaust, Afn=False,
                                             area=1, volume=1, **kwargs)
```

Zone Ventilation Settings.

Setting	Value	Units
Infiltration	☒	
Infiltration rate	0.35	ACH
Natural ventilation	☐	
Nat vent min outdoor air temp	20	degC
Nat vent max outdoor air temp	26	degC
Nat vent max rel humidity	80	
Nat vent schedule	B_Off_Y_Nat	
Nat vent zone temperature setpoint	22	degC
Scheduled ventilation	☐	
Scheduled ventilation ACH	0.6	ACH
Scheduled ventilation schedule	B_Off_Y_Nat	
Scheduled ventilation setpoint	22	degC
Buoyancy	☒	
Wind	☐	
Afn	☐	

Initialize a new VentilationSetting (for zone) object.

Parameters

- **NatVentSchedule** (*UmiSchedule*) – The name of the schedule (Day | Week | Year) which ultimately modifies the Opening Area value. In its current implementation, any value greater

than 0 will consider an open window.

- **ScheduledVentilationSchedule** (*UmiSchedule*, *optional*) – The name of the schedule (Schedules Tab) that modifies the maximum design volume flow rate. This fraction is between 0.0 and 1.0.
- **Afn** (*bool*) – Todo: Not Used.
- **Infiltration** (*float*) – Infiltration rate in ACH.
- **IsBuoyancyOn** (*bool*) – If True, simulation takes into account the stack effect in the infiltration calculation
- **IsInfiltrationOn** (*bool*) – If yes, there is heat transfer between the building and the outside caused by infiltration.
- **IsNatVentOn** (*bool*) – If True, Natural ventilation (air movement/exchange as a result of openings in the building façade not consuming any fan energy).
- **IsScheduledVentilationOn** (*bool*) – If True, Ventilation (flow of air from the outdoor environment directly into a thermal zone) is ON
- **IsWindOn** (*bool*) – If True, simulation takes into account the wind effect in the infiltration calculation
- **NatVentMaxOutdoorAirTemp** (*float*) – The outdoor temperature (in Celsius) above which ventilation is shut off. The minimum value for this field is -100.0°C and the maximum value is 100.0°C. The default value is 100.0°C if the field is left blank. This upper temperature limit is intended to avoid overheating a space, which could result in a cooling load.
- **NatVentMaxRelHumidity** (*float*) – Defines the dehumidifying relative humidity setpoint, expressed as a percentage (0-100), for each timestep of the simulation.
- **NatVentMinOutdoorAirTemp** (*float*) – The outdoor temperature (in Celsius) below which ventilation is shut off. The minimum value for this field is -100.0°C and the maximum value is 100.0°C. The default value is -100.0°C if the field is left blank. This lower temperature limit is intended to avoid overcooling a space, which could result in a heating load.
- **NatVentZoneTempSetpoint** (*float*) –
- **ScheduledVentilationAch** (*float*) – This factor, along with the Zone Volume, will be used to determine the Design Flow Rate.
- **ScheduledVentilationSetpoint** (*float*) – The indoor temperature (in Celsius) below which ventilation is shutoff. The minimum value for this field is -100.0°C and the maximum value is 100.0°C. The default value is -100.0°C if the field is left blank. This lower temperature limit is intended to avoid overcooling a space and thus result in a heating load. For example, if the user specifies a minimum temperature of 20°C, ventilation is assumed to be available if the zone air temperature is above 20°C. If the zone air temperature drops below 20°C, then ventilation is automatically turned off.
- **VentilationType** (*int*) – This alpha character string defines the type of ventilation as one of the following options: Natural, Exhaust, Intake, or Balanced. Natural ventilation is assumed to be air movement/exchange as a result of openings in the building façade and will not consume any fan energy. Values for fan pressure and efficiency for natural ventilation are ignored. For either Exhaust or Intake, values for fan pressure and efficiency define the fan electric consumption. For Natural and Exhaust ventilation, the conditions of the air entering the space are assumed to be equivalent to outside air conditions. For Intake and Balanced

ventilation, an appropriate amount of fan heat is added to the entering air stream. For Balanced ventilation, both an intake fan and an exhaust fan are assumed to co-exist, both having the same flow rate and power consumption (using the entered values for fan pressure rise and fan total efficiency). Thus, the fan electric consumption for Balanced ventilation is twice that for the Exhaust or Intake ventilation types which employ only a single fan.

- ****kwargs** – keywords passed to the constructor.

property NatVentSchedule

Get or set the natural ventilation schedule.

Hint: This schedule ultimately modifies the Opening Area value.

property ScheduledVentilationSchedule

Get or set the scheduled ventilation schedule.

property Infiltration

Get or set the infiltration air change rate [ach].

property IsInfiltrationOn

Get or set the the infiltration [bool].

property IsBuoyancyOn

Get or set the buoyancy boolean.

property IsNatVentOn

Get or set the natural ventilation [bool].

property IsScheduledVentilationOn

Get or set the scheduled ventilation [bool].

property IsWindOn

Get or set the wind effect [bool].

property NatVentMaxOutdoorAirTemp

Get or set the natural ventilation maximum outdoor air temperature [degC].

property NatVentMaxRelHumidity

Get or set the natural ventilation relative humidity setpoint [%].

property NatVentMinOutdoorAirTemp

Get or set the natural ventilation minimum outdoor air temperature [degC].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append *other* to self. Modify and return self.

Parameters *other* ([UmiBase](#)) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

property NatVentZoneTempSetpoint

Get or set the natural ventilation zone temperature setpoint [degC].

property ScheduledVentilationAch

Get or set the scheduled ventilation air changes per hours [-].

property ScheduledVentilationSetpoint

Get or set the scheduled ventilation setpoint.

property VentilationType

Get or set the ventilation type.

Choices are (<VentilationType.Natural: 0>, <VentilationType.Intake: 1>, <VentilationType.Exhaust: 2>, <VentilationType.Balanced: 3>).

property Afn

Get or set the use of the airflow network [bool].

property area

Get or set the area of the zone [m²].

property volume

Get or set the volume of the zone [m³].

classmethod from_dict(*data*, *schedules*, ***kwargs*)

Create a VentilationSetting from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **schedules** (*dict*) – A dictionary of UmiSchedules with their id as keys.

- ****kwargs** – keywords passed parent constructor.

```
{
  "$id": "162",
  "Afn": false,
  "IsBuoyancyOn": true,
  "Infiltration": 0.35,
  "IsInfiltrationOn": true,
  "IsNatVentOn": false,
  "IsScheduledVentilationOn": false,
  "NatVentMaxRelHumidity": 80.0,
  "NatVentMaxOutdoorAirTemp": 26.0,
  "NatVentMinOutdoorAirTemp": 20.0,
  "NatVentSchedule": {
    "$ref": "151"
  },
  "NatVentZoneTempSetpoint": 22.0,
  "ScheduledVentilationAch": 0.6,
  "ScheduledVentilationSchedule": {
    "$ref": "151"
  },
  "ScheduledVentilationSetpoint": 22.0,
  "IsWindOn": false,
  "Category": "Office Spaces",
  "Comments": null,
  "DataSource": "MIT_SDL",
  "Name": "B_Off_0 ventilation"
}
```

to_dict()

Return VentilationSetting dictionary representation.

combine(*other*, *kwargs*)**

Combine VentilationSetting objects together.

Parameters

- **other** (*VentilationSetting*) –
- **kwargs** – keywords passed to constructor.

Returns the combined VentilationSetting object.

Return type (*VentilationSetting*)

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** –

duplicate()

Get copy of self.

to_epbunch(*idf*, *zone_name*, *opening_area=0.0*)

Convert self to the EpBunches given an idf model, a zone name.

Notes

Note that attr:*IsInfiltrationOn*, attr:*IsScheduledVentilationOn* and attr:*IsNatVentOn* must be *True* for their respective EpBunch objects to be created.

Parameters

- **idf** (*IDF*) – The idf model in which the EpBunch is created.
- **zone_name** (*str*) – The zone name to associate this EpBunch.
- **opening_area** (*float*) – The opening area exposed to outdoors (m2) in a zone.

```

ZONEINFILTRATION:DESIGNFLOWRATE,
Zone Infiltration,          !- Name
Zone 1,                    !- Zone or ZoneList Name
AlwaysOn,                  !- Schedule Name
AirChanges/Hour,           !- Design Flow Rate Calculation Method
,                           !- Design Flow Rate
,                           !- Flow per Zone Floor Area
,                           !- Flow per Exterior Surface Area
0.1,                       !- Air Changes per Hour
1,                          !- Constant Term Coefficient
0,                          !- Temperature Term Coefficient
0,                          !- Velocity Term Coefficient
0;                           !- Velocity Squared Term Coefficient

ZONEVENTILATION:DESIGNFLOWRATE,
Zone 1 Ventilation,        !- Name
Zone 1,                    !- Zone or ZoneList Name
AlwaysOn,                  !- Schedule Name
AirChanges/Hour,           !- Design Flow Rate Calculation Method
,                           !- Design Flow Rate
,                           !- Flow Rate per Zone Floor Area
,                           !- Flow Rate per Person
0.6,                       !- Air Changes per Hour
Exhaust,                   !- Ventilation Type
67,                        !- Fan Pressure Rise
0.7,                       !- Fan Total Efficiency
1,                          !- Constant Term Coefficient
0,                          !- Temperature Term Coefficient
0,                          !- Velocity Term Coefficient
0,                          !- Velocity Squared Term Coefficient
-100,                      !- Minimum Indoor Temperature
,                           !- Minimum Indoor Temperature Schedule Name
100,                       !- Maximum Indoor Temperature
,                           !- Maximum Indoor Temperature Schedule Name
-100,                      !- Delta Temperature
,                           !- Delta Temperature Schedule Name
-100,                      !- Minimum Outdoor Temperature
,                           !- Minimum Outdoor Temperature Schedule Name
100,                       !- Maximum Outdoor Temperature
,                           !- Maximum Outdoor Temperature Schedule Name
40;                         !- Maximum Wind Speed)

```

(continues on next page)

(continued from previous page)

```

ZONEVENTILATION:WINDANDSTACKOPENAREA,
,                                !- Name
,                                !- Zone Name
0,                                !- Opening Area
,                                !- Opening Area Fraction Schedule Name
Autocalculate,                  !- Opening Effectiveness
0,                                !- Effective Angle
0,                                !- Height Difference
Autocalculate,                  !- Discharge Coefficient for Opening
-100,                            !- Minimum Indoor Temperature
,                                !- Minimum Indoor Temperature Schedule Name
100,                             !- Maximum Indoor Temperature
,                                !- Maximum Indoor Temperature Schedule Name
-100,                            !- Delta Temperature
,                                !- Delta Temperature Schedule Name
-100,                            !- Minimum Outdoor Temperature
,                                !- Minimum Outdoor Temperature Schedule Name
100,                             !- Maximum Outdoor Temperature
,                                !- Maximum Outdoor Temperature Schedule Name
40;                             !- Maximum Wind Speed

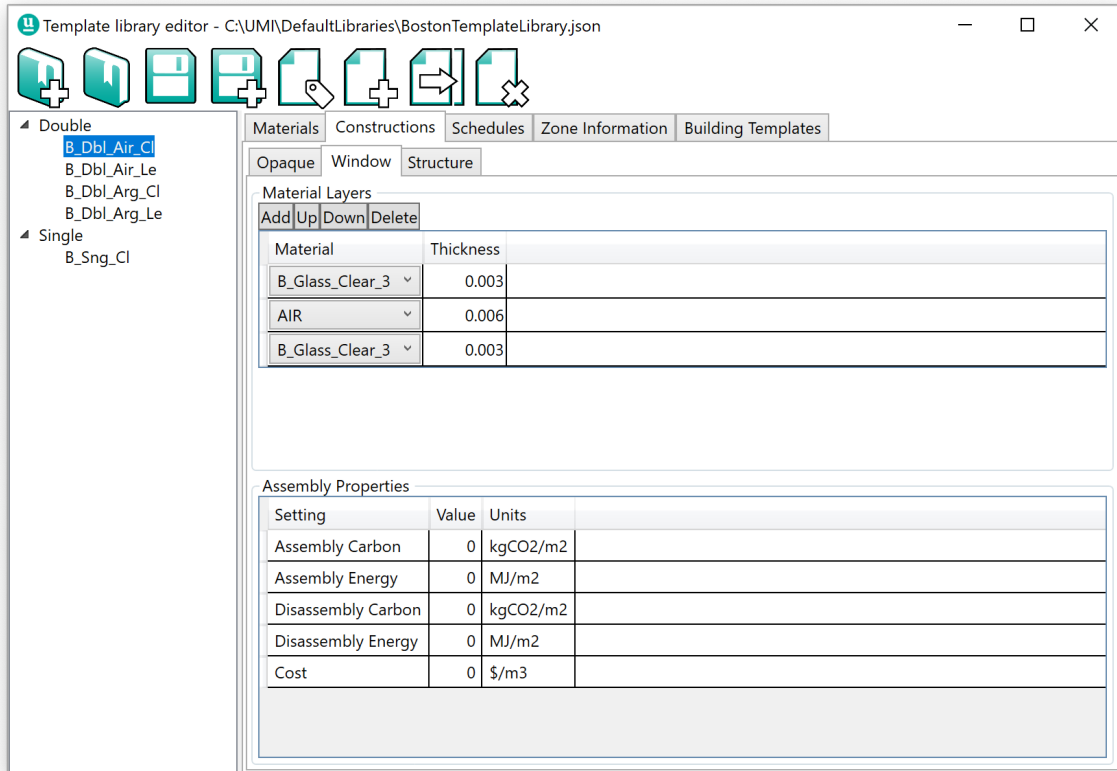
```

Returns A 3-tuple of EpBunch objects added to the idf model.

Return type `tuple`

archetypal.template.WindowConstruction

class archetypal.template.**WindowConstruction**(*Name*, *Layers*, *Category*='Double', ***kwargs*)
Window Construction.



Initialize a WindowConstruction.

Parameters

- **Name** (*str*) – Name of the WindowConstruction.
- **Layers** (*list of (MaterialLayer or GasLayer)*) – List of MaterialLayer and GasLayer.
- **Category** (*str*) – “Single”, “Double” or “Triple”.
- ****kwargs** – Other keywords passed to the constructor.

property Category

Get or set the Category. Choices are (“single”, “double”, “triple”).

property gap_count

Get the number of gas gaps contained within the window construction.

property glazing_count

Get the nb of glazing materials contained within the window construction.

property r_factor

Get the construction R-factor [m2-K/W].

Note: including standard resistances for air films. Formulas for film coefficients come from EN673 / ISO10292.

property r_value

Get or set the thermal resistance [Km2/W] (excluding air films).

property outside_emissivity

Get the hemispherical emissivity of the outside face of the construction.

property inside_emissivity

Get the hemispherical emissivity of the inside face of the construction.

property solar_transmittance

Get the solar transmittance of the window at normal incidence.

property visible_transmittance

Get the visible transmittance of the window at normal incidence.

property thickness

Thickness of the construction [m].

classmethod from_dict(*data*, *materials*, *kwargs*)**

Create an WindowConstruction from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **materials** (*dict*) – A dictionary of materials with their id as keys.
- ****kwargs** – keywords passed to the constructor.

```
data = {
    "$id": "57",
    "Layers": [
        {"Material": {"$ref": "7"}, "Thickness": 0.003},
        {"Material": {"$ref": "1"}, "Thickness": 0.006},
        {"Material": {"$ref": "7"}, "Thickness": 0.003},
    ],
    "AssemblyCarbon": 0.0,
    "AssemblyCost": 0.0,
    "AssemblyEnergy": 0.0,
    "DisassemblyCarbon": 0.0,
    "DisassemblyEnergy": 0.0,
    "Category": "Double",
    "Comments": "default",
    "DataSource": "default",
    "Name": "B_Dbl_Air_Cl",
}
```

classmethod from_epbunch(*Construction*, *kwargs*)**

Create *WindowConstruction* object from idf *Construction* object.

Example

```
>>> from archetypal import IDF
>>> from archetypal.template.window_setting import WindowSetting
>>> idf = IDF("myidf.idf")
>>> construction_name = "Some construction name"
>>> WindowConstruction.from_epbunch(Name=construction_name, idf=idf)
```

Parameters

- **Construction** (*EpBunch*) – The Construction epbunch object.
- ****kwargs** – Other keywords passed to the constructor.

classmethod from_shgc(*Name*, *solar_heat_gain_coefficient*, *u_factor*, *visible_transmittance*=None, ***kwargs*)

Create a WindowConstruction from shgc, u_factor and visible_transmittance.

Parameters

- **Name** (*str*) – The name of the window construction.
- **shgc** (*double*) – The window’s Solar Heat Gain Coefficient.
- **u_factor** (*double*) – The window’s U-value.
- **visible_transmittance** (*double*, *optional*) – The window’s visible transmittance. If none, the visible transmittance defaults to the solar transmittance *t_sol*.
- **kwargs** – keywords passed to the parent constructor.

Returns:

to_dict()

Return WindowConstruction dictionary representation.

to_epbunch(*idf*)

Convert self to a *Construction* epbunch given an idf model.

Parameters **idf** (*IDF*) – The idf model in which the EpBunch is created.

Construction,	
B_Dbl_Air_Cl,	!- Name
B_Glass_Clear_3_0.003_B_Dbl_Air_Cl,	!- Outside Layer
AIR_0.006_B_Dbl_Air_Cl,	!- Layer 2
B_Glass_Clear_3_0.003_B_Dbl_Air_Cl;	!- Layer 3

Returns The EpBunch object added to the idf model.

Return type EpBunch

mapping(*validate*=True)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

combine(*other*, *weights*=None)

Append other to self. Return self + other as a new object.

For now, simply returns self.

Todo:

- Implement equivalent window layers for constant u-factor.
-

validate()

Validate object and fill in missing values.

Todo:

- Implement validation
-

duplicate()

Get copy of self.

temperature_profile(*outside_temperature=-18, inside_temperature=21, wind_speed=6.7, height=1.0, angle=90.0, pressure=101325*)

Get a list of temperatures at each material boundary across the construction.

Parameters

- **outside_temperature** – The temperature on the outside of the construction [C]. Default is -18, which is consistent with NFRC 100-2010.
- **inside_temperature** – The temperature on the inside of the construction [C]. Default is 21, which is consistent with NFRC 100-2010.
- **wind_speed** – The average outdoor wind speed [m/s]. This affects outdoor convective heat transfer coefficient. Default is 6.7 m/s.
- **height** – An optional height for the surface in meters. Default is 1.0 m.
- **angle** – An angle in degrees between 0 and 180. 0 = A horizontal surface with the outside boundary on the bottom. 90 = A vertical surface 180 = A horizontal surface with the outside boundary on the top.
- **pressure** – The average pressure of in Pa. Default is 101325 Pa for standard pressure at sea level.

Returns

A tuple with two elements - temperatures: A list of temperature values [C].

The first value will always be the outside temperature and the second will be the exterior surface temperature. The last value will always be the inside temperature and the second to last will be the interior surface temperature.

- **r_values**: A list of R-values for each of the material layers [m²-K/W]. The first value will always be the resistance of the exterior air and the last value is the resistance of the interior air. The sum of this list is the R-factor for this construction given the input parameters.

shgc(*environmental_conditions='summer', global_radiation=783*)

Calculate the shgc given environmental conditions.

Notes

This method implements a heat balance at each interface of the glazing unit including outside and inside air film resistances, solar radiation absorption in the glass. See `heat_balance()` for more details.

Parameters

- **environmental_conditions** (*str*) – “summer” or “winter”. A window shgc is usually calculated with summer conditions. Default is “summer”.
- **global_radiation** (*float*) – Incident solar radiation [W / m²]. Overwrite the solar radiation used in the calculation of the shgc.

Returns The shgc of the window construction for the given environmental conditions.

Return type *float*

property AssemblyCarbon

Get or set the assembly carbon [kgCO₂/m²].

property AssemblyCost

Get or set the assembly cost [\$ / m²].

property AssemblyEnergy

Get or set the assembly energy [MJ/m2].

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property DisassemblyCarbon

Get or set the disassembly carbon [kgCO2/m2].

property DisassemblyEnergy

Get or set the disassembly energy [MJ/m2].

property Layers:

`List[Union[archetypal.template.materials.material_layer.MaterialLayer,
archetypal.template.materials.gas_layer.GasLayer]]`

Get or set the material layers.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

heat_balance(*environmental_conditions='summer'*, *G_t=783*)

Return heat flux and temperatures at each surface of the window.

Note: Only implemented for glazing with two layers.

Parameters

- **environmental_conditions** (*str*) – The environmental conditions from the NRFC standard used to calculate the heat balance. Default is “summer”.
- **G_t** (*float*) – The incident radiation.

Returns heat_flux, temperature_profile

Return type tuple

property id

Get or set the id.

in_h(*t_kelvin*=293.15, *delta_t*=15, *height*=1.0, *angle*=90, *pressure*=101325)

Get the detailed indoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

in_h_c(*t_kelvin*=293.15, *delta_t*=15, *height*=1.0, *angle*=90, *pressure*=101325)

Get detailed indoor convective heat transfer coef. according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

in_h_simple()

Get the simple indoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

out_h(*wind_speed*=6.7, *t_kelvin*=273.15)

Get the detailed outdoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **wind_speed** (*float*) – The average outdoor wind speed [m/s]. This affects the convective heat transfer coefficient. Default is 6.7 m/s.

- **t_kelvin**(*float*) – The average between the outdoor temperature and the exterior surface temperature. This can affect the radiative heat transfer. Default is 273.15K (0C).

out_h_simple()

Get the simple outdoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

to_ref()

Return a ref pointer to self.

property u_factor

Get the overall heat transfer coefficient (including air films) W/(m2K).

property u_value

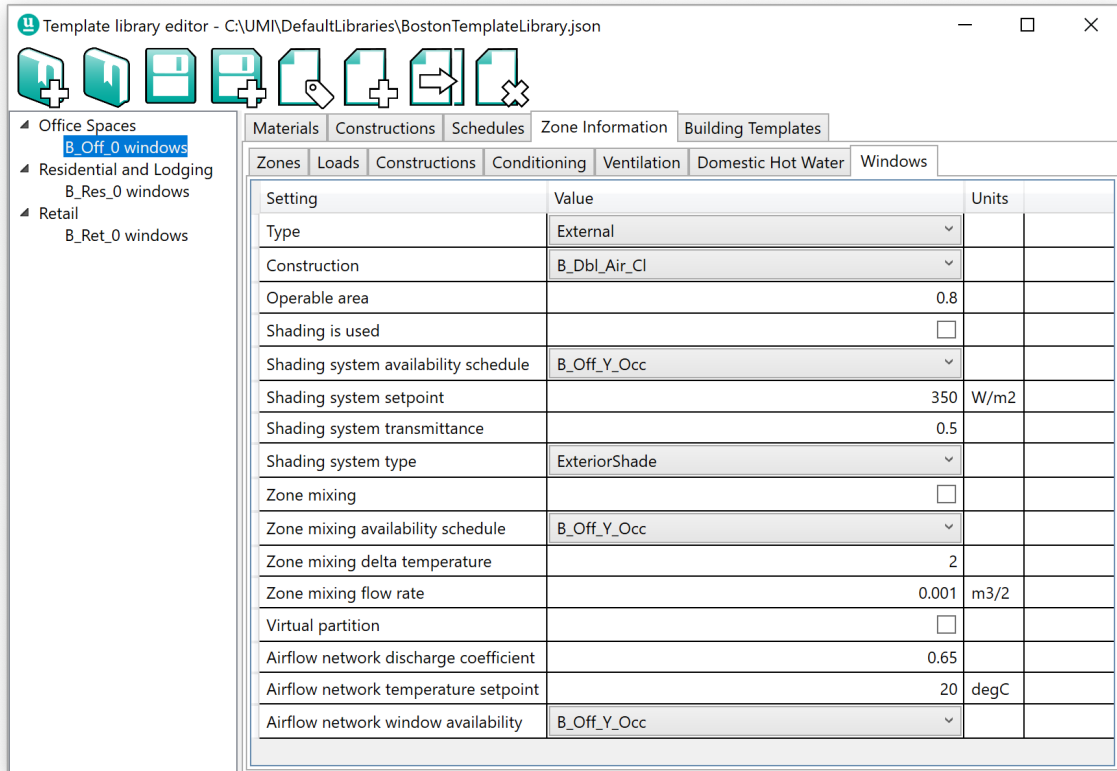
Get the heat transfer coefficient [W/m2K] (excluding air films).

archetypal.template.WindowSetting

```
class archetypal.template.WindowSetting(Name, Construction=None, OperableArea=0.8,
                                         AfnWindowAvailability=None, AfnDischargeC=0.65,
                                         AfnTempSetpoint=20, IsVirtualPartition=False,
                                         IsShadingSystemOn=False,
                                         ShadingSystemAvailabilitySchedule=None,
                                         ShadingSystemSetpoint=180, ShadingSystemTransmittance=0.5,
                                         ShadingSystemType=ShadingType.ExteriorShade,
                                         Type=WindowType.External, IsZoneMixingOn=False,
                                         ZoneMixingAvailabilitySchedule=None,
                                         ZoneMixingDeltaTemperature=2, ZoneMixingFlowRate=0.001,
                                         area=1, **kwargs)
```

Defines the various window-related properties of a Zone.

Control natural ventilation, shading and airflow networks and more using this class. This class serves the same role as the ZoneInformation>Windows tab in the UMI TemplateEditor.



Hint: The WindowSetting class implements two constructors that are tailored to the `eppy` scripting language:

- `from_construction()` and
- `from_surface()`.

Initialize a WindowSetting using default values.

Parameters

- **Construction** (`WindowConstruction`) – The window construction.
- **OperableArea** (`float`) – The operable window area as a ratio of total window area. eg. 0.8 := 80% of the windows area is operable.
- **AfnWindowAvailability** (`UmiSchedule`) – The Airflow Network availability schedule.
- **AfnDischargeC** (`float`) – Airflow Network Discharge Coefficient. Default = 0.65.
- **AfnTempSetpoint** (`float`) – Airflow Network Temperature Setpoint. Default = 20 degreeC.
- **IsVirtualPartition** (`bool`) – Virtual Partition.
- **IsShadingSystemOn** (`bool`) – Shading is used. Default is False.
- **ShadingSystemAvailabilitySchedule** (`UmiSchedule`) – Shading system availability schedule.
- **ShadingSystemSetpoint** (`float`) – Shading system setpoint in units of W/m2. Default = 180 W/m2.

- **ShadingSystemTransmittance** (*float*) – Shading system transmittance. Default = 0.5.
- **ShadingSystemType** (*int*) – Shading System Type. 0 = ExteriorShade, 1 = InteriorShade.
- **Type** (*int*) –
- **IsZoneMixingOn** (*bool*) – Zone mixing.
- **ZoneMixingAvailabilitySchedule** (*UmiSchedule*) – Zone mixing availability schedule.
- **ZoneMixingDeltaTemperature** (*float*) – Zone mixing delta
- **ZoneMixingFlowRate** (*float*) – Zone mixing flow rate in units of m3/m2. Default = 0.001 m3/m2.
- ****kwargs** – other keywords passed to the constructor.

property area

Get or set the area of the zone associated to this object [m²].

property OperableArea

Get or set the operable area ratio [-].

property AfnDischargeC

Get or set the air flow network discharge coefficient.

property AfnTempSetpoint

Get or set the air flow network setpoint temperature [degC].

property AfnWindowAvailability

Get or set the air flow network window availability schedule.

property ShadingSystemType

Get or set the shading system type [enum].

property ShadingSystemSetpoint

Get or set the shading system setpoint [W/m2].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property ShadingSystemTransmittance

Get or set the shading system transmittance [-].

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

property ShadingSystemAvailabilitySchedule

Get or set the shading system availability schedule.

property IsShadingSystemOn

Get or set the use of the shading system.

property ZoneMixingAvailabilitySchedule

Get or set the zone mixing availability schedule.

property ZoneMixingDeltaTemperature

Get or set the zone mixing delta temperature.

property Construction

Get or set the window construction.

property IsVirtualPartition

Get or set the state of the virtual partition.

property IsZoneMixingOn

Get or set mixing in zone.

property ZoneMixingFlowRate

Get or set the zone mixing flow rate [m3/s].

property Type

Get or set the window type [enum].

classmethod generic(*Name*)

Initialize a generic window with SHGC=0.704, UFactor=2.703, Tvis=0.786.

Parameters **Name** (*str*) – Name of the WindowSetting

classmethod from_construction(*Construction*, ***kwargs*)

Make a *WindowSetting* directly from a *Construction* object.

Examples

```
>>> from archetypal import IDF
>>> from archetypal.template.window_setting import WindowSetting
>>> # Given an IDF object
>>> idf = IDF("idfname.idf")
>>> constr = idf.getobject('CONSTRUCTION',
>>>                        'AEDG-SmOffice 1A Window Fixed')
>>> WindowSetting.from_construction(
>>>     Name='test_window',
>>>     Construction=constr
>>> )
```

Parameters

- **Construction** (*EpBunch*) – The construction name for this window.
- ****kwargs** – Other keywords passed to the constructor.

Returns The window setting object.

Return type (*windowSetting*)

classmethod `from_surface(surface, **kwargs)`

Build a *WindowSetting* object from a *FenestrationSurface:Detailed*.

This constructor will detect common window constructions and shading devices. Supported Shading and Natural Air flow EnergyPlus objects are: *WindowProperty:ShadingControl*, *AirflowNetwork:MultiZone:Surface*.

Important: If an EnergyPlus object is not supported, eg.: *AirflowNetwork:MultiZone:Component:DetailedOpening*, only a warning will be issued in the console for the related object instance and default values will be automatically used.

Parameters **surface** (*EpBunch*) – The *FenestrationSurface:Detailed* object.

Returns The window setting object.

Return type (*WindowSetting*)

combine(*other, weights=None, allow_duplicates=False*)

Append other to self. Return self + other as a new object.

Parameters

- **other** (*WindowSetting*) – The other *OpaqueMaterial* object
- **weights** (*list-like, optional*) – A list-like object of len 2. If None, equal weights are used.

Returns A new combined object made of self + other.

Return type *WindowSetting*

to_dict()

Return *WindowSetting* dictionary representation.

classmethod `from_dict(data, schedules, window_constructions, **kwargs)`

Create a *ZoneConditioning* from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **schedules** (*dict*) – A dictionary of UmiSchedules with their id as keys.
- **window_constructions** (*dict*) – A dictionary of WindowConstruction objects with their id as keys.
- ****kwargs** – keywords passed to parent constructor.

classmethod **from_ref**(*ref*, *building_templates*, *schedules*, *window_constructions*, ***kwargs*)
Initialize *WindowSetting* object from a reference id.

Hint: In some cases, the WindowSetting is referenced in the DataStore to the Windows property of a BuildingTemplate (instead of being listed in the WindowSettings list. This is the case in the original BostonTemplateLibrary.json.

Parameters

- **ref** (*str*) – The referenced number in the json library.
- **building_templates** (*list*) – List of BuildingTemplates from the datastore.

Returns The parsed WindowSetting.

Return type *WindowSetting*

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

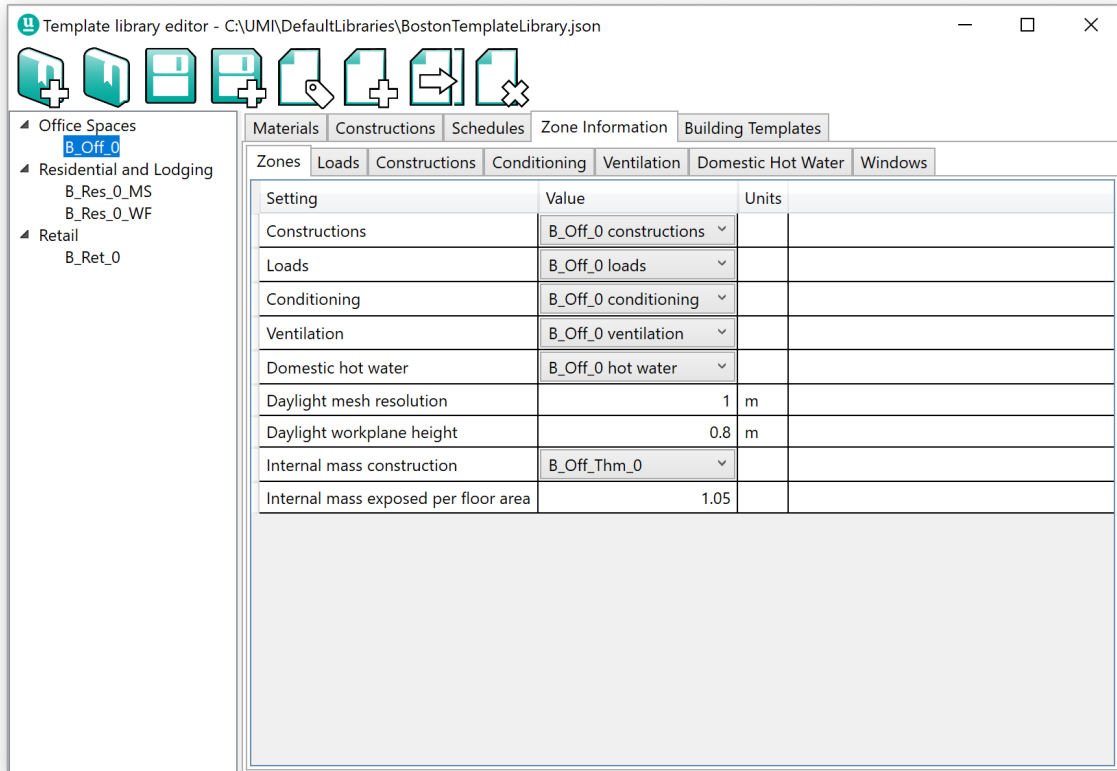
Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

archetypal.template.ZoneDefinition

```
class archetypal.template.ZoneDefinition(Name, Constructions=None, Loads=None,  
                                         Conditioning=None, Ventilation=None,  
                                         DomesticHotWater=None, DaylightMeshResolution=1,  
                                         DaylightWorkplaneHeight=0.8,  
                                         InternalMassConstruction=None,  
                                         InternalMassExposedPerFloorArea=1.05, Windows=None,  
                                         area=1, volume=1, occupants=1,  
                                         is_part_of_conditioned_floor_area=True,  
                                         is_part_of_total_floor_area=True, multiplier=1,  
                                         zone_surfaces=None, is_core=False, **kwargs)
```

Zone settings class.



Initialize Zone object.

Parameters

- **Name** (*str*) – Name of the object. Must be Unique.
- **Constructions** (*ZoneConstructionSet*) –
- **Loads** (*ZoneLoad*) – Loads of the zone defined with the lights, equipment and occupancy parameters (see *ZoneLoad*)
- **Conditioning** (*ZoneConditioning*) – Conditioning of the zone defined with heating/cooling and mechanical ventilation parameters (see *ZoneConditioning*)
- **Ventilation** (*VentilationSetting*) – Ventilation settings of the zone defined with the infiltration rate and natural ventilation parameters (see *VentilationSetting*)
- **DomesticHotWater** (*archetypal.template.dhw.DomesticHotWaterSetting*) –
- **DaylightMeshResolution** (*float*) –
- **DaylightWorkplaneHeight** (*float*) –
- **InternalMassConstruction** (*archetypal.OpaqueConstruction*) –
- **InternalMassExposedPerFloorArea** –
- **Windows** (*WindowSetting*) – The WindowSetting object associated with this zone.
- **area** (*float*) –
- **volume** (*float*) –
- **occupants** (*float*) –

- ****kwargs** –

property Constructions

Get or set the ZoneConstructionSet object.

property Loads

Get or set the ZoneLoad object.

property Conditioning

Get or set the ZoneConditioning object.

property Ventilation

Get or set the VentilationSetting object.

property DomesticHotWater

Get or set the DomesticHotWaterSetting object.

property DaylightMeshResolution

Get or set the daylight mesh resolution [m].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* ([UmiBase](#)) –

Returns self

Return type [UmiBase](#)

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** ([UmiBase](#)) – The other UmiBase object to calculate average value with.
- **attr** ([str](#)) – The attribute of the UmiBase object.
- **weights** ([iterable](#), *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

to_ref()

Return a ref pointer to self.

property DaylightWorkplaneHeight

Get or set the DaylightWorkplaneHeight [m].

property InternalMassConstruction

Get or set the internal mass construction object.

property InternalMassExposedPerFloorArea

Get or set the internal mass exposed per floor area [-].

property Windows

Get or set the WindowSetting object.

property occupants

Get or set the number of occupants in the zone.

property area

Get or set the area of the zone [m²].

property volume

Get or set the volume of the zone [m³].

property is_core

Get or set if the zone is a core zone [bool].

property multiplier

Get or set the zone multiplier.

Note: Zone multiplier is designed as a “multiplier” for floor area, zone loads, and energy consumed by internal gains.

property is_part_of_conditioned_floor_area

Get or set is part of conditioned area [bool].

property is_part_of_total_floor_area

Get or set is part of the total building floor area [bool].

property zone_surfaces

Get or set the list of surfaces for this zone.

to_dict()

Return ZoneDefinition dictionary representation.

classmethod from_dict(*data*, *zone_conditionings*, *zone_construction_sets*, *domestic_hot_water_settings*, *opaque_constructions*, *zone_loads*, *ventilation_settings*, *kwargs*)**

Create a ZoneDefinition from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **zone_conditionings** (*dict*) – A dictionary of ZoneConditioning objects with their id as keys.

- **zone_construction_sets** (*dict*) – A dictionary of ZoneConstructionSet objects with their id as keys.
- **domestic_hot_water_settings** (*dict*) – A dictionary of DomesticHotWaterSetting objects with their id as keys.
- **opaque_constructions** (*dict*) – A dictionary of OpaqueConstruction objects with their id as keys.
- **zone_loads** (*dict*) – A dictionary of ZoneLoad objects with their id as keys.
- **ventilation_settings** (*dict*) – A dictionary of ZoneConditioning objects with their id as keys.
- ****kwargs** – keywords passed to the constructor.

```
{
  "$id": "175",
  "Conditioning": {
    "$ref": "165"
  },
  "Constructions": {
    "$ref": "168"
  },
  "DaylightMeshResolution": 1.0,
  "DaylightWorkplaneHeight": 0.8,
  "DomesticHotWater": {
    "$ref": "159"
  },
  "InternalMassConstruction": {
    "$ref": "54"
  },
  "InternalMassExposedPerFloorArea": 1.05,
  "Loads": {
    "$ref": "172"
  },
  "Ventilation": {
    "$ref": "162"
  },
  "Category": "Office Spaces",
  "Comments": null,
  "DataSource": "MIT_SDL",
  "Name": "B_Off_0"
}
```

classmethod from_epbunch(*ep_bunch*, *construct_parents=True*, ***kwargs*)

Create a Zone object from an eppy ‘ZONE’ epbunch.

Parameters

- **ep_bunch** (*eppy.bunch_subclass.EpBunch*) – The Zone EpBunch.
- **construct_parents** (*bool*) – If False, skips construction of parents objects such as Constructions, Conditioning, etc.

combine(*other*, *weights=None*, *allow_duplicates=False*)

Combine two ZoneDefinition objects together.

Parameters

- **other** (*ZoneDefinition*) – The other object.
- **weights** (*list-like, optional*) – A list-like object of len 2. If None, the volume of the zones for which self and other belongs is used.

Todo: Create Equivalent InternalMassConstruction from partitions when combining zones.

Returns the combined Zone object.

Return type (*ZoneDefinition*)

validate()

Validate object and fill in missing values.

mapping (*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

archetypal.template.ZoneConstructionSet

```
class archetypal.template.ZoneConstructionSet(Name, Facade=None, Ground=None, Partition=None,
                                              Roof=None, Slab=None, IsFacadeAdiabatic=False,
                                              IsGroundAdiabatic=False, IsPartitionAdiabatic=False,
                                              IsRoofAdiabatic=False, IsSlabAdiabatic=False, area=1,
                                              volume=1, **kwargs)
```

ZoneConstructionSet class.

Create a ZoneConstructionSet object.

Parameters

- **Name** (*str*) – Name of the object. Must be Unique.
- **Facade** (*OpaqueConstruction*) – The OpaqueConstruction object representing a facade.
- **Ground** (*OpaqueConstruction*) – The OpaqueConstruction object representing a ground floor.
- **Partition** (*OpaqueConstruction*) – The OpaqueConstruction object representing a partition wall.
- **Roof** (*OpaqueConstruction*) – The OpaqueConstruction object representing a roof.
- **Slab** (*OpaqueConstruction*) – The OpaqueConstruction object representing a slab.
- **IsFacadeAdiabatic** (*bool*) – If True, surface is adiabatic.
- **IsGroundAdiabatic** (*bool*) – If True, surface is adiabatic.
- **IsPartitionAdiabatic** (*bool*) – If True, surface is adiabatic.
- **IsRoofAdiabatic** (*bool*) – If True, surface is adiabatic.
- **IsSlabAdiabatic** (*bool*) – If True, surface is adiabatic.
- ****kwargs** –

property Facade

Get or set the Facade OpaqueConstruction.

property Ground

Get or set the Ground OpaqueConstruction.

property Partition

Get or set the Partition OpaqueConstruction.

property Roof

Get or set the Roof OpaqueConstruction.

property Slab

Get or set the Slab OpaqueConstruction.

property IsFacadeAdiabatic

Get or set is facade adiabatic [bool].

property IsGroundAdiabatic

Get or set is ground adiabatic [bool].

property IsPartitionAdiabatic

Get or set is partition adiabatic [bool].

property IsRoofAdiabatic

Get or set is roof adiabatic [bool].

property IsSlabAdiabatic

Get or set is slab adiabatic [bool].

property area

Get or set the area of the zone [m²].

property volume

Get or set the volume of the zone [m³].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** ([UmiBase](#)) –

Returns self

Return type [UmiBase](#)

float_mean(*other*, *attr*, *weights*=None)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** ([UmiBase](#)) – The other UmiBase object to calculate average value with.
- **attr** ([str](#)) – The attribute of the UmiBase object.

- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

classmethod from_dict (*data*, *opaque_constructions*, ***kwargs*)

Create a ZoneConstructionSet from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **opaque_constructions** (*dict*) – A dictionary of OpaqueConstruction with their id as keys.
- ****kwargs** – keywords passed parent constructor.

```
{
  "$id": "168",
  "Facade": {
    "$ref": "35"
  },
  "Ground": {
    "$ref": "42"
  },
  "Partition": {
    "$ref": "48"
  },
  "Roof": {
    "$ref": "39"
  },
  "Slab": {
    "$ref": "45"
  },
  "IsFacadeAdiabatic": false,
  "IsGroundAdiabatic": false,
  "IsPartitionAdiabatic": false,
  "IsRoofAdiabatic": false,
  "IsSlabAdiabatic": false,
  "Category": "Office Spaces",
  "Comments": null,
  "DataSource": "MIT_SDL",
  "Name": "B_Off_0 constructions"
}
```

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename (*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_ref()

Return a ref pointer to self.

combine(*other*, *weights=None*, ***kwargs*)

Combine two ZoneConstructionSet objects together.

Parameters

- **other** (*ZoneConstructionSet*) –
- **kwargs** – keywords passed to constructor.

Returns the combined ZoneConstructionSet object.

Return type (*ZoneConstructionSet*)

to_dict()

Return ZoneConstructionSet dictionary representation.

validate()

Validate object and fill in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

duplicate()

Get copy of self.

1.12.4 Template Helper Classes

Classes that support the *Template Classes* classes above.

<i>umi_base.UmiBase</i>	Base class for template objects.
<i>materials.material_base.MaterialBase</i>	A class used to store data linked with the Life Cycle aspect of materials.
<i>materials.material_layer.MaterialLayer</i>	Class used to define one layer in a construction assembly.
<i>constructions.base_construction.ConstructionBase</i>	A class used to store data linked to Life Cycle aspects.
<i>constructions.base_construction.LayeredConstruction</i>	Defines the layers of an <i>OpaqueConstruction</i> .
<i>structure.MassRatio</i>	Handles the properties of the mass ratio for building template structure.
<i>schedule.YearSchedulePart</i>	Helper Class for YearSchedules defined with FromDay FromMonth ToDay ToMonth.
<i>schedule.DaySchedule</i>	Superclass of UmiSchedule that handles daily schedules.
<i>schedule.WeekSchedule</i>	Superclass of UmiSchedule that handles weekly schedules.
<i>schedule.YearSchedule</i>	Superclass of UmiSchedule that handles yearly schedules.
<i>constructions.window_construction.WindowType</i>	Refers to the window type.
<i>constructions.window_construction.ShadingType</i>	Refers to window shading types.

archetypal.template.umi_base.UmiBase

```
class archetypal.template.umi_base.UmiBase(Name, Category='Uncategorized', Comments='',
                                         DataSource=None, allow_duplicates=False, **kwargs)
```

Base class for template objects.

The UmiBase class handles common properties to all Template objects.

Parameters

- **Name** (*str*) – Unique, the name of the object.
- **Category** (*str*) – Group objects by assigning the same category identifier. Thies can be any string.
- **Comments** (*str*) – A comment displayed in the UmiTemplateLibrary.
- **DataSource** (*str*) – A description of the datasource of the object. This helps identify from which data is the current object created.
- **allow_duplicates** (*bool*) – If True, this object can be equal to another one if it has a different name.
- ****kwargs** –

property Name

Get or set the name of the object.

property id

Get or set the id.

property DataSource

Get or set the datasource of the object.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property allow_duplicates

Get or set the use of duplicates [bool].

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

duplicate()

Get copy of self.

rename(name)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_dict()

Return UmiBase dictionary representation.

to_ref()

Return a ref pointer to self.

float_mean(other, attr, weights=None)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

extend(*other, allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –**Returns** self**Return type** *UmiBase***validate**()

Validate UmiObjects and fills in missing values.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.**get_unique**()

Return first object matching equality in the list of instantiated objects.

archetypal.template.materials.material_base.MaterialBase

```
class archetypal.template.materials.material_base.MaterialBase(Name, Cost=0,
                                                                EmbodiedCarbon=0,
                                                                EmbodiedEnergy=0,
                                                                SubstitutionTimestep=100,
                                                                TransportCarbon=0,
                                                                TransportDistance=0,
                                                                TransportEnergy=0,
                                                                SubstitutionRatePattern=None,
                                                                **kwargs)
```

A class used to store data linked with the Life Cycle aspect of materials.

For more information on the Life Cycle Analysis performed in UMI, see: <https://umidocs.readthedocs.io/en/latest/docs/life-cycle-introduction.html#life-cycle-impact>

Initialize a MaterialBase object with parameters.

Parameters

- **Name** (*str*) – Name of the Material.
- **Cost** (*float*) – The purchase cost of the material by volume (\$/m3).
- **EmbodiedCarbon** (*float*) – Represents the GHG emissions through the lifetime of the product (kgCO2/kg).
- **EmbodiedEnergy** (*float*) – Represents all fuel consumption (Typically from non-renewable sources) which happened through the lifetime of a product (or building), expressed as primary energy (MJ/kg).
- **SubstitutionTimestep** (*float*) – The duration in years of a period of replacement (e.g. There will be interventions in this material type every 10 years).

- **TransportCarbon** (*float*) – The impacts associated with the transport by km of distance and kg of material (kgCO2/kg/km).
- **TransportDistance** (*float*) – The average distance in km from the manufacturing site to the building construction site
- **TransportEnergy** (*float*) – The impacts associated with the transport by km of distance and kg of material (MJ/kg/km).
- **SubstitutionRatePattern** (*list-like*) – A ratio from 0 to 1 which defines the amount of the material replaced at the end of each period of replacement, *SubstitutionTimestep* (e.g. Every 10 years this cladding will be completely replaced with ratio 1). Notice that you can define different replacement ratios for different consecutive periods, introducing them separated by commas. For example, if you introduce the series “0.1 , 0.1 , 1” after the first 10 years a 10% will be replaced, then after 20 years another 10%, then after 30 years a 100%, and finally the series would start again in year 40.
- ****kwargs** – Keywords passed to the UmiBase class. See UmiBase for more details.

property Cost

Get or set the cost of the material [\$].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property EmbodiedCarbon

Get or set the embodied carbon of the material [kgCO2/kg].

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_dict()

Return UmiBase dictionary representation.

to_ref()

Return a ref pointer to self.

property EmbodiedEnergy

Get or set the embodied energy of the material [MJ/kg].

property SubstitutionTimestep

Get or set the substitution timestep of the material.

property SubstitutionRatePattern

Get or set the substitution rate pattern of the material.

property TransportCarbon

Get or set the transportation carbon of the material [kgCO2/kg/km].

property TransportDistance

Get or set the transportation distance of the material [km].

property TransportEnergy

Get or set the transportation energy of the material [MJ/kg/km].

validate()

Validate object and fill in missing values.

archetypal.template.materials.material_layer.MaterialLayer

class archetypal.template.materials.material_layer.**MaterialLayer**(*Material, Thickness, **kwargs*)

Class used to define one layer in a construction assembly.

This class has two attributes:

1. **Material** (*OpaqueMaterial* or *GlazingMaterial* or *GasMaterial*): the material object for this layer.
2. **Thickness** (*float*): The thickness of the material in the layer.

Initialize a MaterialLayer object with parameters.

Parameters

- **Material** (*OpaqueMaterial, GlazingMaterial, GasMaterial*) –
- **Thickness** (*float*) – The thickness of the material in the construction.

property Material

Get or set the material of self.

property Thickness

Get or set the material thickness [m].

property resistivity

Get or set the resistivity of the material layer [m-K/W].

property r_value

Get or set the the R-value of the material layer [m²-K/W].

Note that, when setting the R-value, the thickness of the material will be adjusted and the conductivity will remain fixed.

property u_value

Get or set the heat transfer coefficient [W/(m²K)].

property heat_capacity

Get the material layer's heat capacity [J/(m²-k)].

property specific_heat

Get the material layer's specific heat [J/kg-K].

to_dict()

Return MaterialLayer dictionary representation.

to_epbunch(idf)

Convert self to an EpBunch given an IDF model.

Notes

The object is added to the idf model. The thickness is passed to the epbunch.

Parameters **idf** (**IDF**) – An IDF model.

Returns The EpBunch object added to the idf model.

Return type EpBunch

mapping()

Get a dict based on the object properties, useful for dict repr.

get_unique()

Return the first of all the created objects that is equivalent to self.

duplicate()

Get copy of self.

archetypal.template.constructions.base_construction.ConstructionBase

```
class archetypal.template.constructions.base_construction.ConstructionBase(Name, Assembly-  
Carbon=0, AssemblyCost=0, AssemblyEn-  
ergy=0, DisassemblyCar-  
bon=0, DisassemblyEn-  
ergy=0, **kwargs)
```

A class used to store data linked to Life Cycle aspects.

For more information on the Life Cycle Analysis performed in UMI, see: <https://umidocs.readthedocs.io/en/latest/docs/life-cycle-introduction.html#life-cycle-impact>

Initialize a ConstructionBase object with parameters.

Parameters

- **AssemblyCarbon** (*float*) – assembly carbon [kgCO2/m2].
- **AssemblyCost** (*float*) – assembly carbon [kgCO2/m2].
- **AssemblyEnergy** (*float*) – assembly energy [MJ/m2].
- **DisassemblyCarbon** (*float*) – disassembly carbon [kgCO2/m2].
- **DisassemblyEnergy** (*float*) – disassembly energy [MJ/m2].
- ****kwargs** – keywords passed to UmiBase.

property AssemblyCarbon

Get or set the assembly carbon [kgCO2/m2].

property AssemblyCost

Get or set the assembly cost [\$/m2].

property AssemblyEnergy

Get or set the assembly energy [MJ/m2].

property DisassemblyCarbon

Get or set the disassembly carbon [kgCO2/m2].

property DisassemblyEnergy

Get or set the disassembly energy [MJ/m2].

validate()

Validate object and fill in missing values.

duplicate()

Get copy of self.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters *other* (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters *validate* (*bool*) – If True, try to validate object before returning the mapping.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters *name* (*str*) – the name.

to_dict()

Return UmiBase dictionary representation.

to_ref()

Return a ref pointer to self.

archetypal.template.constructions.base_construction.LayeredConstruction

class archetypal.template.constructions.base_construction.**LayeredConstruction**(*Name*, *Layers*,
***kwargs*)

Defines the layers of an OpaqueConstruction.

Layers

List of MaterialLayer objects from outside to inside.

Type list of archetypal.MaterialLayer

Initialize Layered Construction.

Parameters

- **Layers** (*list of* ([MaterialLayer](#) or [GasLayer](#))) – A list of [MaterialLayer](#) or [GasLayer](#) objects.
- ****kwargs** – Keywords passed to the [ConstructionBase](#) constructor.

property Layers:

`List[Union[archetypal.template.materials.material_layer.MaterialLayer,
archetypal.template.materials.gas_layer.GasLayer]]`

Get or set the material layers.

property r_value

Get or set the thermal resistance [Km2/W] (excluding air films).

property AssemblyCarbon

Get or set the assembly carbon [kgCO2/m2].

property AssemblyCost

Get or set the assembly cost [\$/m2].

property AssemblyEnergy

Get or set the assembly energy [MJ/m2].

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property DisassemblyCarbon

Get or set the disassembly carbon [kgCO2/m2].

property DisassemblyEnergy

Get or set the disassembly energy [MJ/m2].

property Name

Get or set the name of the object.

property allow_duplicates

Get or set the use of duplicates [bool].

duplicate()

Get copy of self.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** ([UmiBase](#)) –

Returns self

Return type [UmiBase](#)

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** ([UmiBase](#)) – The other [UmiBase](#) object to calculate average value with.
- **attr** (*str*) – The attribute of the [UmiBase](#) object.

- **weights**(*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

get_unique()

Return first object matching equality in the list of instantiated objects.

property id

Get or set the id.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

rename(*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

to_dict()

Return UmiBase dictionary representation.

to_ref()

Return a ref pointer to self.

property u_value

Get the heat transfer coefficient [W/m²K] (excluding air films).

validate()

Validate object and fill in missing values.

property r_factor

Get the R-factor [m²-K/W] (including air films).

out_h_simple()

Get the simple outdoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

in_h_simple()

Get the simple indoor heat transfer coefficient according to ISO 10292.

This is used for all opaque R-factor calculations.

out_h(*wind_speed=6.7*, *t_kelvin=273.15*)

Get the detailed outdoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **wind_speed** (*float*) – The average outdoor wind speed [m/s]. This affects the convective heat transfer coefficient. Default is 6.7 m/s.
- **t_kelvin** (*float*) – The average between the outdoor temperature and the exterior surface temperature. This can affect the radiative heat transfer. Default is 273.15K (0C).

in_h(*t_kelvin=293.15*, *delta_t=15*, *height=1.0*, *angle=90*, *pressure=101325*)

Get the detailed indoor heat transfer coefficient according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

in_h_c(*t_kelvin=293.15, delta_t=15, height=1.0, angle=90, pressure=101325*)

Get detailed indoor convective heat transfer coef. according to ISO 15099.

This is used for window U-factor calculations and all of the temperature_profile calculations.

Parameters

- **t_kelvin** (*float*) – The average between the indoor temperature and the interior surface temperature. Default is 293.15K (20C).
- **delta_t** (*float*) – The temperature difference between the indoor temperature and the interior surface temperature [C]. Default is 15C.
- **height** (*float*) – An optional height for the surface in meters. Default is 1.0 m, which is consistent with NFRC standards.
- **angle** (*float*) – An angle in degrees between 0 and 180. 0 = A horizontal surface with downward heat flow through the layer. 90 = A vertical surface 180 = A horizontal surface with upward heat flow through the layer.
- **pressure** (*float*) – The average pressure in Pa. Default is 101325 Pa for standard pressure at sea level.

property outside_emissivity

Get the hemispherical emissivity of the outside face of the construction.

property inside_emissivity

Get the emissivity of the inside face of the construction [-].

property u_factor

Get the overall heat transfer coefficient (including air films) W/(m2K).

archetypal.template.structure.MassRatio

```
class archetypal.template.structure.MassRatio(HighLoadRatio=None, Material=None,  
                                              NormalRatio=None, **kwargs)
```

Handles the properties of the mass ratio for building template structure.

Initialize a MassRatio object with parameters.

Parameters

- **HighLoadRatio** (*float*) –
- **Material** (*OpaqueMaterial*) –

- **NormalRatio** (*float*) –

property HighLoadRatio
Get or set the high load ratio [kg/m2].

property Material
Get or set the structure OpaqueMaterial.

property NormalRatio
Get or set the normal load ratio [kg/m2].

to_dict()
Return MassRatio dictionary representation.

mapping()
Get a dict based on the object properties, useful for dict repr.

get_unique()
Return the first of all the created objects that is equivalent to self.

classmethod generic()
Create generic MassRatio object.

duplicate()
Get copy of self.

archetypal.template.schedule.YearSchedulePart

class archetypal.template.schedule.**YearSchedulePart**(*FromDay=None, FromMonth=None, ToDay=None, ToMonth=None, Schedule=None, **kwargs*)

Helper Class for YearSchedules defined with FromDay FromMonth ToDay ToMonth.

Initialize YearSchedulePart.

Parameters

- **FromDay** (*int*) – This numeric field is the starting day for the schedule time period.
- **FromMonth** (*int*) – This numeric field is the starting month for the schedule time period.
- **ToDay** (*int*) – This numeric field is the ending day for the schedule time period.
- **ToMonth** (*int*) – This numeric field is the ending month for the schedule time period.
- **Schedule** (*UmiSchedule*) – The associated UmiSchedule related to this object.
- **kwargs** (*dict*) – Other Keyword arguments.

property FromDay
Get or set the start day-of-month number [int].

property FromMonth
Get or set the start month-number [int].

property ToDay
Get or set the end day-of-month number [int].

property ToMonth
Get or set the end month-number [int].

property Schedule
Get or set the WeekSchedule object.

classmethod `from_dict(data, schedules, **kwargs)`
Create a YearSchedulePart object from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **schedules** (*dict*) – A dictionary of WeekSchedules with their id as keys.
- ****kwargs** – keywords passed to parent constructor.

```
data = {
    'FromDay': 1,
    'FromMonth': 1,
    'ToDay': 31,
    'ToMonth': 12,
    'Schedule': {'$ref': '140622440042800'}
}
```

to_dict()
Return YearSchedulePart dictionary representation.

mapping()
Get a dict based on the object properties, useful for dict repr.

get_unique()
Return the first of all the created objects that is equivalent to self.

archetypal.template.schedule.DaySchedule

class `archetypal.template.schedule.DaySchedule(Name, Values, Category='Day', **kwargs)`
Superclass of UmiSchedule that handles daily schedules.

Initialize a DaySchedule object with parameters.

Parameters

- **Values** (*list*) – List of 24 values.
- **Name** (*str*) – Name of the schedule.
- **Category** (*str*) – category identification (default: “Day”).
- ****kwargs** – Keywords passed to the *UmiSchedule* constructor.

property `all_values: numpy.ndarray`
Return numpy array of schedule Values.

classmethod `from_epbunch(epbunch, strict=False, **kwargs)`
Create a DaySchedule from an EpBunch.

This method accepts “Schedule:Day:Hourly”, “Schedule:Day:List” and “Schedule:Day:Interval”.

Parameters

- **epbunch** (*EpBunch*) – The EpBunch object to construct a DaySchedule from.
- ****kwargs** – Keywords passed to the *UmiSchedule* constructor. See *UmiSchedule* for more details.

classmethod `from_values(Name, Values, Type='Fraction', **kwargs)`
Create a DaySchedule from an array of size (24,).

Parameters

- **Name** –
- **Values** (*array-like*) – A list of values of length 24.
- **Type** (*str*) – Schedule Type Limit name.
- ****kwargs** – Keywords passed to the *UmiSchedule* constructor. See *UmiSchedule* for more details.

classmethod from_dict(*data*, ***kwargs*)

Create a DaySchedule from a dictionary.

Parameters

- **data** (*dict*) – A python dictionary with the structure shown bellow.
- ****kwargs** – keywords passed to parents constructors.

```
{
    "$id": "67",
    "Category": "Day",
    "Type": "Fraction",
    "Values": [...], # 24 hourly values
    "Comments": "default",
    "DataSource": "default",
    "Name": "B_Res_D_Occ_WD"
},
```

get_unique()

Return the first of all the created objects that is equivalent to self.

to_dict()

Return DaySchedule dictionary representation.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

to_ref()

Return a ref pointer to self.

duplicate()

Get copy of self.

to_epbunch(*idf*)

Convert self to an epbunch given an idf model.

Parameters **idf** (*IDF*) – An IDF model.

```
SCHEDULE:DAY:HOURLY,
,                                !- Name
,                                !- Schedule Type Limits Name
0,                                !- Hour 1
0,                                !- Hour 2
0,                                !- Hour 3
0,                                !- Hour 4
0,                                !- Hour 5
0,                                !- Hour 6
```

(continues on next page)

(continued from previous page)

```
0,                !- Hour 7
0,                !- Hour 8
0,                !- Hour 9
0,                !- Hour 10
0,                !- Hour 11
0,                !- Hour 12
0,                !- Hour 13
0,                !- Hour 14
0,                !- Hour 15
0,                !- Hour 16
0,                !- Hour 17
0,                !- Hour 18
0,                !- Hour 19
0,                !- Hour 20
0,                !- Hour 21
0,                !- Hour 22
0,                !- Hour 23
0;                !- Hour 24
```

Returns The EpBunch object added to the idf model.

Return type EpBunch

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the schedule.

property Type

Get or set the schedule type limits object. Can be None.

property Values

Get or set the list of schedule values.

property allow_duplicates

Get or set the use of duplicates [bool].

combine(*other*, *weights=**None*, *quantity=**None*)

Combine two UmiSchedule objects together.

Parameters

- **other** (*UmiSchedule*) – The other Schedule object to combine with.
- **weights** (*list*, *dict* or *string*) – Attribute of self and other containing the weight factor. If a list is passed, it must have len = 2; the first element is applied to self and the second element is applied to other. If a dict is passed, the self.Name and other.Name are the keys. If a str is passed, the

- **quantity** (*list or dict or bool*) – Scalar value that will be multiplied by self before the averaging occurs. This ensures that the resulting schedule returns the correct integrated value. If a dict is passed, keys are schedules Names and values are quantities.

Returns the combined UmiSchedule object.

Return type (*UmiSchedule*)

Raises **TypeError** – if Quantity is not of type list, tuple, dict or a callable.

classmethod **constant_schedule**(*value=1, Name='AlwaysOn', Type='Fraction', **kwargs*)
Create an UmiSchedule with a constant value at each timestep.

Parameters

- **Type** –
- **value** (*float*) –
- **Name** –
- **idf** –
- ****kwargs** –

develop()

Develop the UmiSchedule into a Year-Week-Day schedule structure.

extend(*other, allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other, attr, weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

get_ref(*ref*)

Get item matching reference id.

Parameters **ref** –

static **get_schedule_type_limits_name**(*epbunch*)

Return the Schedule Type Limits name associated to this schedule.

property **id**

Get or set the id.

property **max**

Get the maximum value of the schedule.

property **mean**

Get the mean value of the schedule.

property **min**

Get the minimum value of the schedule.

plot(**kwargs)

Plot the schedule. Implements the .loc accessor on the series object.

Notes

Plotting can also be achieved through the series property: *Schedule.series.plot()*.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF()
>>> epbunch = idf.schedules_dict["NECB-A-Thermostat Setpoint-Heating"]
>>> s = Schedule.from_epbunch(epbunch)
>>> )
>>> s.plot(drawstyle="steps-post")
```

Parameters ****kwargs** (*dict*) – keyword arguments passed to `EnergySeries.plot()`.

plot2d(**kwargs)

Plot the carpet plot of the schedule. Plot rectangular data as a color-encoded 2d-matrix.

Parameters

- **periodlength** (*int*) – The period length to show on the yaxis. By default, the period length will be calculated to represent one day of data on the yaxis and days on the xaxis.
- **vmin** (*float*) – The data value that defines 0.0 in the normalization. Defaults to the min value of the dataset.
- **vmax** (*float*) – The data value that defines 1.0 in the normalization. Defaults to the the max value of the dataset.
- **vcenter** (*float*) – The data value that defines 0.5 in the normalization.
- **axis_off** (*bool*) – If True, no axis is plotted.
- **cmap** (*str* or *Colormap*) – Colormap to select colors from. If string, load colormap with that name from matplotlib.
- **figsize** (*tuple*) – Size of a figure object. A tuple (width, height) in inches.
- **show** (*bool*) – whether to display the figure or not.
- **save** (*bool*) – whether to save the figure to disk or not.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg').
- **colorbar** (*bool*) – If True, plot the colorbar.
- **ax** (*Axes*) – An axes of the current figure.
- **filename** (*string*) – the name of the file to save.
- **extent** (*str* or *Bbox*) – Bounding box in inches: only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- **ylabel** (*str*) – Set the label for the y-axis.

- **xlabel** (*str*) – Set the label for the x-axis.
- **ax_title** (*bool* or *str*) – Title to use for the ax. If True, the Series name is used.
- ****kwargs** – Options to pass to matplotlib imshow method.

Returns

tuple containing:

fig: is the Matplotlib Figure object ax: a single axis object.

Return type (*tuple*)**property predecessors**

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

property quantity

Get or set the schedule quantity.

classmethod random (*Name='AlwaysOn', Type='Fraction', **kwargs*)

Create an UmiSchedule with a randomized value (0-1) at each timestep.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Type** (*str* or *ScheduleTypeLimits*) –
- ****kwargs** – keywords passed to the constructor.

rename (*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

replace (*new_values: pandas.core.series.Series*)

Replace values with new values while keeping the full load hours constant.

Time steps that are not specified in *new_values* will be adjusted to keep the full load hours of the schedule constant. No check whether the new schedule stays between the bounds set by self.Type is done. Be aware.

scale (*diversity=0.1*)

Scale the schedule values by a diversity factor around the average.

property series

Return an EnergySeries.

property startDate

Get the start date of the schedule. Satisfies *startDayOfTheWeek*.

to_year_week_day()

Convert to three-tuple epbunch given an idf model.

Returns ‘Schedule:Year’, ‘Schedule:Week:Daily’ and ‘Schedule:Day:Hourly’ representations.

Returns

3-element tuple containing

- **yearly** (*Schedule*): The yearly schedule object
- **weekly** (*list of Schedule*): The list of weekly schedule objects
- **daily** (*list of Schedule*): The list of daily schedule objects

validate()

Validate object and fill in missing values.

archetypal.template.schedule.WeekSchedule

class archetypal.template.schedule.**WeekSchedule**(*Name, Days=None, Category='Week', **kwargs*)

Superclass of UmiSchedule that handles weekly schedules.

Initialize a WeekSchedule object with parameters.

Parameters

- **Days** (*list of DaySchedule*) – list of [DaySchedule](#).
- ****kwargs** –

property Days

Get or set the list of DaySchedule objects.

classmethod **from_epbunch**(*epbunch, **kwargs*)

Create a WeekSchedule from a Schedule:Week:Daily object.

Parameters

- **epbunch** (*EpBunch*) – The Schedule:Week:Daily object.
- ****kwargs** – keywords passed to the constructor.

classmethod **from_dict**(*data, day_schedules, **kwargs*)

Create a WeekSchedule from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **day_schedules** (*dict*) – A dictionary of python DaySchedules with their id as keys.
- ****kwargs** – keywords passed to the constructor.

get_unique()

Return the first of all the created objects that is equivalent to self.

to_dict()

Return WeekSchedule dictionary representation.

mapping(*validate=True*)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

classmethod **get_days**(*epbunch, **kwargs*)

Get the DaySchedules referenced in the Week:Schedule:Days object.

Parameters **DaySchedule** (*list of*) – The list of DaySchedules referenced by the epbunch.

property **all_values:** **numpy.ndarray**

Return numpy array of schedule Values.

to_ref()

Return a ref pointer to self.

duplicate()

Get copy of self.

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the schedule.

property Type

Get or set the schedule type limits object. Can be None.

property Values

Get or set the list of schedule values.

property allow_duplicates

Get or set the use of duplicates [bool].

combine(*other*, *weights=None*, *quantity=None*)

Combine two UmiSchedule objects together.

Parameters

- **other** (*UmiSchedule*) – The other Schedule object to combine with.
- **weights** (*list*, *dict* or *string*) – Attribute of self and other containing the weight factor. If a list is passed, it must have len = 2; the first element is applied to self and the second element is applied to other. If a dict is passed, the self.Name and other.Name are the keys. If a str is passed, the
- **quantity** (*list* or *dict* or *bool*) – Scalar value that will be multiplied by self before the averaging occurs. This ensures that the resulting schedule returns the correct integrated value. If a dict is passed, keys are schedules Names and values are quantities.

Returns the combined UmiSchedule object.

Return type (*UmiSchedule*)

Raises *TypeError* – if Quantity is not of type list, tuple, dict or a callable.

classmethod constant_schedule(*value=1*, *Name='AlwaysOn'*, *Type='Fraction'*, ***kwargs*)

Create an UmiSchedule with a constant value at each timestep.

Parameters

- **Type** –
- **value** (*float*) –
- **Name** –
- **idf** –
- ****kwargs** –

develop()

Develop the UmiSchedule into a Year-Week-Day schedule structure.

extend(*other*, *allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns `self`

Return type *UmiBase*

float_mean(*other*, *attr*, *weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable*, *optional*) – Weights of [self, other] to calculate weighted average.

classmethod from_values(*Name*, *Values*, *Type='Fraction'*, ***kwargs*)

Create an UmiSchedule from a list of values.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Values** (*list*) –
- **Type** –
- ****kwargs** –

get_ref(*ref*)

Get item matching reference id.

Parameters **ref** –

static get_schedule_type_limits_name(*epbunch*)

Return the Schedule Type Limits name associated to this schedule.

property id

Get or set the id.

property max

Get the maximum value of the schedule.

property mean

Get the mean value of the schedule.

property min

Get the minimum value of the schedule.

plot(***kwargs*)

Plot the schedule. Implements the .loc accessor on the series object.

Notes

Plotting can also be acheived through the series property: *Schedule.series.plot()*.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF()
>>> epbunch = idf.schedules_dict["NECB-A-Thermostat Setpoint-Heating"]
>>> s = Schedule.from_epbunch(epbunch)
>>> )
>>> s.plot(drawstyle="steps-post")
```

Parameters ****kwargs** (*dict*) – keyword arguments passed to `EnergySeries.plot()`.

`plot2d(**kwargs)`

Plot the carpet plot of the schedule. Plot rectangular data as a color-encoded 2d-matrix.

Parameters

- **periodlength** (*int*) – The period length to show on the yaxis. By default, the period length will be calculated to represent one day of data on the yaxis and days on the xaxis.
- **vmin** (*float*) – The data value that defines 0.0 in the normalization. Defaults to the min value of the dataset.
- **vmax** (*float*) – The data value that defines 1.0 in the normalization. Defaults to the the max value of the dataset.
- **vcenter** (*float*) – The data value that defines 0.5 in the normalization.
- **axis_off** (*bool*) – If True, no axis is plotted.
- **cmap** (*str* or *Colormap*) – Colormap to select colors from. If string, load colormap with that name from matplotlib.
- **figsize** (*tuple*) – Size of a figure object. A tuple (width, height) in inches.
- **show** (*bool*) – whether to display the figure or not.
- **save** (*bool*) – whether to save the figure to disk or not.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg').
- **colorbar** (*bool*) – If True, plot the colorbar.
- **ax** (*Axes*) – An axes of the current figure.
- **filename** (*string*) – the name of the file to save.
- **extent** (*str* or *Bbox*) – Bounding box in inches: only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- **ylabel** (*str*) – Set the label for the y-axis.
- **xlabel** (*str*) – Set the label for the x-axis.
- **ax_title** (*bool* or *str*) – Title to use for the ax. If True, the Series name is used.
- ****kwargs** – Options to pass to matplotlib imshow method.

Returns

tuple containing:

fig: is the Matplotlib Figure object ax: a single axis object.

Return type (tuple)

property predecessors

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

property quantity

Get or set the schedule quantity.

classmethod random(Name='AlwaysOn', Type='Fraction', **kwargs)

Create an UmiSchedule with a randomized value (0-1) at each timestep.

Parameters

- **Name** (str) – The name of the Schedule.
- **Type** (str or ScheduleTypeLimits) –
- ****kwargs** – keywords passed to the constructor.

rename(name)

renames self as well as the cached object

Parameters **name** (str) – the name.

replace(new_values: pandas.core.series.Series)

Replace values with new values while keeping the full load hours constant.

Time steps that are not specified in *new_values* will be adjusted to keep the full load hours of the schedule constant. No check whether the new schedule stays between the bounds set by self.Type is done. Be aware.

scale(diversity=0.1)

Scale the schedule values by a diversity factor around the average.

property series

Return an EnergySeries.

property startDate

Get the start date of the schedule. Satisfies *startDayOfTheWeek*.

to_epbunch(idf)

Convert self to an epbunch given an idf model.

Parameters **idf** (IDF) – An IDF model.

Returns The EpBunch object added to the idf model.

Return type EpBunch

to_year_week_day()

Convert to three-tuple epbunch given an idf model.

Returns 'Schedule:Year', 'Schedule:Week:Daily' and 'Schedule:Day:Hourly' representations.

Returns

3-element tuple containing

- **yearly** (Schedule): The yearly schedule object
- **weekly** (list of Schedule): The list of weekly schedule objects
- **daily** (list of Schedule): The list of daily schedule objects

validate()

Validate object and fill in missing values.

archetypal.template.schedule.YearSchedule

class archetypal.template.schedule.**YearSchedule**(*Name*, *Type*='Fraction', *Parts*=None, *Category*='Year', ***kwargs*)

Superclass of UmiSchedule that handles yearly schedules.

Initialize a YearSchedule object with parameters.

Parameters

- **Category** –
- **Name** –
- **Type** –
- **Parts** (*list of YearSchedulePart*) – The YearScheduleParts.
- ****kwargs** –

property Category

Get or set the Category attribute.

property Comments

Get or set the object comments.

property DataSource

Get or set the datasource of the object.

property Name

Get or set the name of the schedule.

property Type

Get or set the schedule type limits object. Can be None.

property Values

Get or set the list of schedule values.

property allow_duplicates

Get or set the use of duplicates [bool].

combine(*other*, *weights*=None, *quantity*=None)

Combine two UmiSchedule objects together.

Parameters

- **other** (*UmiSchedule*) – The other Schedule object to combine with.
- **weights** (*list, dict or string*) – Attribute of self and other containing the weight factor. If a list is passed, it must have len = 2; the first element is applied to self and the second element is applied to other. If a dict is passed, the self.Name and other.Name are the keys. If a str is passed, the
- **quantity** (*list or dict or bool*) – Scalar value that will be multiplied by self before the averaging occurs. This ensures that the resulting schedule returns the correct integrated value. If a dict is passed, keys are schedules Names and values are quantities.

Returns the combined UmiSchedule object.

Return type (*UmiSchedule*)

Raises **TypeError** – if Quantity is not of type list, tuple, dict or a callable.

classmethod constant_schedule(*value=1, Name='AlwaysOn', Type='Fraction', **kwargs*)
Create an UmiSchedule with a constant value at each timestep.

Parameters

- **Type** –
- **value** (*float*) –
- **Name** –
- **idf** –
- ****kwargs** –

develop()

Develop the UmiSchedule into a Year-Week-Day schedule structure.

duplicate()

Get copy of self.

extend(*other, allow_duplicates*)

Append other to self. Modify and return self.

Parameters **other** (*UmiBase*) –

Returns self

Return type *UmiBase*

float_mean(*other, attr, weights=None*)

Calculates the average attribute value of two floats. Can provide weights.

Parameters

- **other** (*UmiBase*) – The other UmiBase object to calculate average value with.
- **attr** (*str*) – The attribute of the UmiBase object.
- **weights** (*iterable, optional*) – Weights of [self, other] to calculate weighted average.

classmethod from_epbunch(*epbunch, strict=False, Type=None, **kwargs*)

Create a Schedule from an epbunch.

Parameters

- **epbunch** –
- **strict** –
- ****kwargs** –

classmethod from_values(*Name, Values, Type='Fraction', **kwargs*)

Create an UmiSchedule from a list of values.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Values** (*list*) –
- **Type** –
- ****kwargs** –

get_ref(*ref*)

Get item matching reference id.

Parameters ref –

static `get_schedule_type_limits_name(epbunch)`

Return the Schedule Type Limits name associated to this schedule.

property `id`

Get or set the id.

property `max`

Get the maximum value of the schedule.

property `mean`

Get the mean value of the schedule.

property `min`

Get the minimum value of the schedule.

plot(***kwargs*)

Plot the schedule. Implements the `.loc` accessor on the series object.

Notes

Plotting can also be achieved through the series property: `Schedule.series.plot()`.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF()
>>> epbunch = idf.schedules_dict["NECB-A-Thermostat Setpoint-Heating"]
>>> s = Schedule.from_epbunch(epbunch)
>>>
>>> s.plot(drawstyle="steps-post")
```

Parameters ***kwargs* (*dict*) – keyword arguments passed to `EnergySeries.plot()`.

plot2d(***kwargs*)

Plot the carpet plot of the schedule. Plot rectangular data as a color-encoded 2d-matrix.

Parameters

- **periodlength** (*int*) – The period length to show on the yaxis. By default, the period length will be calculated to represent one day of data on the yaxis and days on the axis.
- **vmin** (*float*) – The data value that defines 0.0 in the normalization. Defaults to the min value of the dataset.
- **vmax** (*float*) – The data value that defines 1.0 in the normalization. Defaults to the the max value of the dataset.
- **vcenter** (*float*) – The data value that defines 0.5 in the normalization.
- **axis_off** (*bool*) – If True, no axis is plotted.
- **cmap** (*str* or *Colormap*) – Colormap to select colors from. If string, load colormap with that name from matplotlib.
- **figsize** (*tuple*) – Size of a figure object. A tuple (width, height) in inches.
- **show** (*bool*) – whether to display the figure or not.

- **save** (*bool*) – whether to save the figure to disk or not.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg').
- **colorbar** (*bool*) – If True, plot the colorbar.
- **ax** (*Axes*) – An axes of the current figure.
- **filename** (*string*) – the name of the file to save.
- **extent** (*str* or *Bbox*) – Bounding box in inches: only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- **ylabel** (*str*) – Set the label for the y-axis.
- **xlabel** (*str*) – Set the label for the x-axis.
- **ax_title** (*bool* or *str*) – Title to use for the ax. If True, the Series name is used.
- ****kwargs** – Options to pass to matplotlib imshow method.

Returns

tuple containing:

fig: is the Matplotlib Figure object ax: a single axis object.

Return type (*tuple*)**property predecessors**

Get or set the predecessors of self.

Of which objects is self made of. If from nothing else then self, return self.

property quantity

Get or set the schedule quantity.

classmethod random (*Name='AlwaysOn', Type='Fraction', **kwargs*)

Create an UmiSchedule with a randomized value (0-1) at each timestep.

Parameters

- **Name** (*str*) – The name of the Schedule.
- **Type** (*str* or *ScheduleTypeLimits*) –
- ****kwargs** – keywords passed to the constructor.

rename (*name*)

renames self as well as the cached object

Parameters **name** (*str*) – the name.

replace (*new_values: pandas.core.series.Series*)

Replace values with new values while keeping the full load hours constant.

Time steps that are not specified in *new_values* will be adjusted to keep the full load hours of the schedule constant. No check whether the new schedule stays between the bounds set by self.Type is done. Be aware.

scale (*diversity=0.1*)

Scale the schedule values by a diversity factor around the average.

property series

Return an EnergySeries.

property startDate

Get the start date of the schedule. Satisfies *startDayOfTheWeek*.

to_year_week_day()

Convert to three-tuple epbunch given an idf model.

Returns ‘Schedule:Year’, ‘Schedule:Week:Daily’ and ‘Schedule:Day:Hourly’ representations.

Returns

3-element tuple containing

- **yearly** (*Schedule*): The yearly schedule object
- **weekly** (*list of Schedule*): The list of weekly schedule objects
- **daily** (*list of Schedule*): The list of daily schedule objects

validate()

Validate object and fill in missing values.

property all_values: numpy.ndarray

Return numpy array of schedule Values.

classmethod from_dict(data, week_schedules, **kwargs)

Create a YearSchedule from a dictionary.

Parameters

- **data** (*dict*) – The python dictionary.
- **week_schedule** (*dict*) – A dictionary of python WeekSchedules with their id as keys.
- ****kwargs** – keywords passed to the constructor.

get_unique()

Return the first of all the created objects that is equivalent to self.

to_dict()

Return YearSchedule dictionary representation.

to_epbunch(idf)

Convert self to an epbunch given an idf model.

Notes

The object is added to the idf model.

Parameters **idf** (*IDF*) – An IDF model.

Returns The EpBunch object added to the idf model.

Return type EpBunch

mapping(validate=True)

Get a dict based on the object properties, useful for dict repr.

Parameters **validate** (*bool*) – If True, try to validate object before returning the mapping.

to_ref()

Return a ref pointer to self.

archetypal.template.constructions.window_construction.WindowType

class archetypal.template.constructions.window_construction.WindowType(*value*)
Refers to the window type. Two choices are available: interior or exterior.

archetypal.template.constructions.window_construction.ShadingType

class archetypal.template.constructions.window_construction.ShadingType(*value*)
Refers to window shading types.

Hint: EnergyPlus specifies 8 different shading types, but only 2 are supported here: InteriorShade and ExteriorShade. See [shading](#) for more info.

1.12.5 Graph Module

ZoneGraph

A subclass of [networkx.Graph](#).

archetypal.zone_graph.ZoneGraph

class archetypal.zone_graph.ZoneGraph(*incoming_graph_data=None, **attr*)
A subclass of [networkx.Graph](#). This class implements useful methods to visualize and navigate a template along the thermal adjacency of its zones.

There are currently two methods to visualize the graph:

- *plot in 3d* to get a 3-dimensional view of the building.
- *plot in 2d* to get a 2-dimensional view of the building zones

Note: A Graph stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) Python objects with optional key/value attributes. By convention *None* is not used as a node.

Edges are represented as links between nodes with optional key/value attributes.

Initialize a graph with edges, name, or graph attributes.

Wrapper around the [networkx.Graph](#) class.

Parameters

- **incoming_graph_data** – input graph (optional, default: None) Data to initialize graph. If None (default) an empty graph is created. The data can be an edge list, or any NetworkX graph object. If the corresponding optional Python packages are installed the data can also be a NumPy matrix or 2d ndarray, a SciPy sparse matrix, or a PyGraphviz graph.
- **attr** – keyword arguments, optional (default= no attributes) Attributes to add to graph as key=value pairs.

classmethod `from_idf(idf, log_adj_report=True, **kwargs)`

Create a graph representation of all the building zones. An edge between two zones represents the adjacency of the two zones.

If `skeleton` is `False`, this method will create all the building objects iteratively over the building zones.

Parameters

- **log_adj_report** (*bool*, *optional*) – If `True`, prints an adjacency report in the log.
- **skeleton** (*bool*, *optional*) – If `True`, create a zone graph without creating hierarchical objects, eg. `zones > zoneloads > ect.`
- **force** (*bool*) – If `True`, will recalculate the graph.

Returns The building’s zone graph object

Return type *ZoneGraph*

plot_graph3d(*fig_height=None, fig_width=6, save=False, show=True, close=False, ax=None, axis_off=False, cmap='plasma', dpi=300, file_format='png', azimuth=-60, elev=30, proj_type='persp', filename=None, annotate=False, plt_style='ggplot'*)

Plot the `archetypal.template.ZoneGraph` in a 3D plot.

The size of the node is relative to its `networkx.Graph.degree()`. The node degree is the number of edges adjacent to the node.

The nodes are positioned in 3d space according to the mean value of the surfaces centroids. For concave volumes, this corresponds to the center of gravity of the volume. Some weird positioning can occur for convex volumes.

Todo: Create an Example

Parameters

- **fig_height** (*float*) – matplotlib figure height in inches.
- **fig_width** (*float*) – matplotlib figure width in inches.
- **save** (*bool*) – if `True`, save the figure as an image file to disk.
- **show** (*bool*) – if `True`, show the figure.
- **close** (*bool*) – close the figure (only if `show` equals `False`) to prevent display.
- **ax** (*matplotlib.axes._axes.Axes*, *optional*) – An existing axes object on which to plot this graph.
- **axis_off** (*bool*) – If `True`, turn off the matplotlib axis.
- **cmap** (*str*) – The name a registered `matplotlib.colors.Colormap`.
- **dpi** (*int*) – the resolution of the image file if saving.
- **file_format** (*str*) – the format of the file to save (e.g., `'jpg'`, `'png'`, `'svg'`, `'pdf'`)
- **azim** (*float*) – Azimuthal viewing angle, defaults to `-60`.
- **elev** (*float*) – Elevation viewing angle, defaults to `30`.
- **proj_type** (*str*) – Type of projection, accepts `'persp'` and `'ortho'`.
- **filename** (*str*) – the name of the file if saving.

- **annotate** (*bool* or *str* or *tuple*) – If True, annotates the node with the Zone Name. Pass an EpBunch *field_name* to retrieve data from the zone EpBunch. Pass a tuple (data, key) to retrieve data from the graph: eg. ('core', None) will retrieve the attribute 'core' associated to the node. The second tuple element serves as a key on the first: `G.nodes(data=data)[key]`.
- **plt_style** (*str*, *dict*, or *list*) – A style specification. Valid options are: - *str*: The name of a style or a path/URL to a style file. For a list of available style names, see `style.available`. - *dict*: Dictionary with valid key/value pairs for `matplotlib.rcParams`. - *list*: A list of style specifiers (*str* or *dict*) applied from first to last in the list.

Returns `fig, ax`

Return type `fig, ax`

plot_graph2d(*layout_function*, **func_args*, *color_nodes=None*, *fig_height=None*, *fig_width=6*, *node_labels_to_integers=False*, *legend=False*, *with_labels=True*, *arrows=True*, *save=False*, *show=True*, *close=False*, *ax=None*, *axis_off=False*, *cmap='plasma'*, *dpi=300*, *file_format='png'*, *filename='unnamed'*, *plt_style='ggplot'*, *extent='tight'*, ***kwargs*)

Plot the adjacency of the zones as a graph. Choose a layout from the `networkx.drawing.layout` module, the `Graphviz AGraph (dot)` module, the `Graphviz with pydot` module. Then, plot the graph using matplotlib using the `networkx.drawing.py_lab`

Examples

```
>>> import networkx as nx
>>> G = BuildingTemplate().from_idf
>>> G.plot_graph2d(nx.nx_agraph.graphviz_layout, ('dot'),
>>>                 font_color='w', legend=True, font_size=8,
>>>                 color_nodes='core',
>>>                 node_labels_to_integers=True,
>>>                 plt_style='seaborn', save=True,
>>>                 filename='test')
```

Parameters

- **layout_function** (*func*) – One of the networkx layout functions.
- ***func_args** – The layout function arguments as a tuple. The first argument (self) is already supplied.
- **color_nodes** (*bool* or *str*) – False by default. If a string is passed the nodes are colored according to a data attribute of the graph. By default, the original node names is accessed with the 'name' attribute.
- **fig_height** (*float*) – matplotlib figure height in inches.
- **fig_width** (*float*) – matplotlib figure width in inches.
- **node_labels_to_integers** –
- **legend** –
- **with_labels** (*bool*, *optional*) – Set to True to draw labels on the
- **arrows** (*bool*, *optional*) – If True, draw arrowheads. Note: Arrows will be the same color as edges.
- **save** (*bool*) – if True, save the figure as an image file to disk.

- **show** (*bool*) – if True, show the figure.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **ax** (*matplotlib.axes._axes.Axes*, *optional*) – An existing axes object on which to plot this graph.
- **axis_off** (*bool*) – If True, turn off the matplotlib axis.
- **cmap** (*str*) – The name a registered `matplotlib.colors.Colormap`.
- **dpi** (*int*) – the resolution of the image file if saving.
- **file_format** (*str*) – the format of the file to save (e.g., ‘jpg’, ‘png’, ‘svg’, ‘pdf’)
- **filename** (*str*) – the name of the file if saving.
- **plt_style** (*str*, *dict*, or *list*) – A style specification. Valid options are: - *str*: The name of a style or a path/URL to a style file. For a list of available style names, see *style.available* . - *dict*: Dictionary with valid key/value pairs for `matplotlib.rcParams`. - *list*: A list of style specifiers (*str* or *dict*) applied from first to last in the list.
- **extent** –
- ****kwargs** – keywords passed to `networkx.draw_networkx()`

Returns The fig and ax objects

Return type (*tuple*)

property core_graph

Returns a copy of the ZoneGraph containing only core zones

property perim_graph

Returns a copy of the ZoneGraph containing only perimeter zones

info(*node=None*)

Print short summary of information for the graph or the node n.

Parameters **node** (*any hashable*) – A node in the graph

add_edge(*u_of_edge*, *v_of_edge*, ***attr*)

Add an edge between u and v.

The nodes u and v will be automatically added if they are not already in the graph.

Edge attributes can be specified with keywords or by directly accessing the edge’s attribute dictionary. See examples below.

Parameters

- **u_of_edge** (*nodes*) – Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) Python objects.
- **v_of_edge** (*nodes*) – Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) Python objects.
- **attr** (*keyword arguments*, *optional*) – Edge data (or labels or objects) can be assigned using keyword arguments.

See also:

[*add_edges_from*](#) add a collection of edges

Notes

Adding an edge that already exists updates the edge data.

Many NetworkX algorithms designed for weighted graphs use an edge attribute (by default *weight*) to hold a numerical value.

Examples

The following all add the edge $e=(1, 2)$ to graph G :

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> e = (1, 2)
>>> G.add_edge(1, 2) # explicit two-node form
>>> G.add_edge(*e) # single edge as tuple of two nodes
>>> G.add_edges_from([(1, 2)]) # add edges from iterable container
```

Associate data to edges using keywords:

```
>>> G.add_edge(1, 2, weight=3)
>>> G.add_edge(1, 3, weight=7, capacity=15, length=342.7)
```

For non-string attribute keys, use subscript notation.

```
>>> G.add_edge(1, 2)
>>> G[1][2].update({0: 5})
>>> G.edges[1, 2].update({0: 5})
```

add_edges_from(*ebunch_to_add*, ***attr*)

Add all the edges in *ebunch_to_add*.

Parameters

- **ebunch_to_add** (*container of edges*) – Each edge given in the container will be added to the graph. The edges must be given as 2-tuples (u, v) or 3-tuples (u, v, d) where d is a dictionary containing edge data.
- **attr** (*keyword arguments, optional*) – Edge data (or labels or objects) can be assigned using keyword arguments.

See also:

add_edge add a single edge

add_weighted_edges_from convenient way to add weighted edges

Notes

Adding the same edge twice has no effect but any edge data will be updated when each duplicate edge is added.

Edge attributes specified in an ebunch take precedence over attributes specified via keyword arguments.

Examples

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_edges_from([(0, 1), (1, 2)]) # using a list of edge tuples
>>> e = zip(range(0, 3), range(1, 4))
>>> G.add_edges_from(e) # Add the path graph 0-1-2-3
```

Associate data to edges

```
>>> G.add_edges_from([(1, 2), (2, 3)], weight=3)
>>> G.add_edges_from([(3, 4), (1, 4)], label="WN2898")
```

add_node(*node_for_adding*, ***attr*)

Add a single node *node_for_adding* and update node attributes.

Parameters

- **node_for_adding** (*node*) – A node can be any hashable Python object except None.
- **attr** (*keyword arguments, optional*) – Set or change node attributes using key=value.

See also:

[add_nodes_from](#)

Examples

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_node(1)
>>> G.add_node("Hello")
>>> K3 = nx.Graph([(0, 1), (1, 2), (2, 0)])
>>> G.add_node(K3)
>>> G.number_of_nodes()
3
```

Use keywords set/change node attributes:

```
>>> G.add_node(1, size=10)
>>> G.add_node(3, weight=0.4, UTM=("13S", 382871, 3972649))
```

Notes

A hashable object is one that can be used as a key in a Python dictionary. This includes strings, numbers, tuples of strings and numbers, etc.

On many platforms hashable items also include mutables such as NetworkX Graphs, though one should be careful that the hash doesn't change on mutables.

add_nodes_from(*nodes_for_adding*, ***attr*)

Add multiple nodes.

Parameters

- **nodes_for_adding** (*iterable container*) – A container of nodes (list, dict, set, etc.).
OR A container of (node, attribute dict) tuples. Node attributes are updated using the attribute dict.

- **attr** (*keyword arguments, optional (default= no attributes)*) – Update attributes for all nodes in nodes. Node attributes specified in nodes as a tuple take precedence over attributes specified via keyword arguments.

See also:

[add_node](#)

Examples

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_nodes_from("Hello")
>>> K3 = nx.Graph([(0, 1), (1, 2), (2, 0)])
>>> G.add_nodes_from(K3)
>>> sorted(G.nodes(), key=str)
[0, 1, 2, 'H', 'e', 'l', 'o']
```

Use keywords to update specific node attributes for every node.

```
>>> G.add_nodes_from([1, 2], size=10)
>>> G.add_nodes_from([3, 4], weight=0.4)
```

Use (node, attrdict) tuples to update attributes for specific nodes.

```
>>> G.add_nodes_from([(1, dict(size=11)), (2, {"color": "blue"})])
>>> G.nodes[1]["size"]
11
>>> H = nx.Graph()
>>> H.add_nodes_from(G.nodes(data=True))
>>> H.nodes[1]["size"]
11
```

add_weighted_edges_from(*ebunch_to_add*, *weight='weight'*, ***attr*)

Add weighted edges in *ebunch_to_add* with specified weight attr

Parameters

- **ebunch_to_add** (*container of edges*) – Each edge given in the list or container will be added to the graph. The edges must be given as 3-tuples (u, v, w) where w is a number.
- **weight** (*string, optional (default= 'weight')*) – The attribute name for the edge weights to be added.
- **attr** (*keyword arguments, optional (default= no attributes)*) – Edge attributes to add/update for all edges.

See also:

[add_edge](#) add a single edge

[add_edges_from](#) add multiple edges

Notes

Adding the same edge twice for Graph/DiGraph simply updates the edge data. For Multi-Graph/MultiDiGraph, duplicate edges are stored.

Examples

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_weighted_edges_from([(0, 1, 3.0), (1, 2, 7.5)])
```

property adj

Graph adjacency object holding the neighbors of each node.

This object is a read-only dict-like structure with node keys and neighbor-dict values. The neighbor-dict is keyed by neighbor to the edge-data-dict. So `G.adj[3][2]['color'] = 'blue'` sets the color of the edge (3, 2) to “blue”.

Iterating over `G.adj` behaves like a dict. Useful idioms include *for nbr, datadict in G.adj[n].items()*.

The neighbor information is also provided by subscripting the graph. So *for nbr, foovalue in G[node].data('foo', default=1)*: works.

For directed graphs, `G.adj` holds outgoing (successor) info.

adjacency()

Returns an iterator over (node, adjacency dict) tuples for all nodes.

For directed graphs, only outgoing neighbors/adjacencies are included.

Returns `adj_iter` – An iterator over (node, adjacency dictionary) for all nodes in the graph.

Return type iterator

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> [(n, nbrdict) for n, nbrdict in G.adjacency()]
[(0, {1: {}}), (1, {0: {}, 2: {}}), (2, {1: {}, 3: {}}), (3, {2: {}})]
```

adjlist_inner_dict_factory

alias of `dict`

adjlist_outer_dict_factory

alias of `dict`

clear()

Remove all nodes and edges from the graph.

This also removes the name, and all graph, node, and edge attributes.

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.clear()
>>> list(G.nodes)
[]
>>> list(G.edges)
[]
```

clear_edges()

Remove all edges from the graph without altering nodes.

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.clear_edges()
>>> list(G.nodes)
[0, 1, 2, 3]
>>> list(G.edges)
[]
```

copy(as_view=False)

Returns a copy of the graph.

The copy method by default returns an independent shallow copy of the graph and attributes. That is, if an attribute is a container, that container is shared by the original and the copy. Use Python's *copy.deepcopy* for new containers.

If *as_view* is True then a view is returned instead of a copy.

Notes

All copies reproduce the graph structure, but data attributes may be handled in different ways. There are four types of copies of a graph that people might want.

Deepcopy – A “deepcopy” copies the graph structure as well as all data attributes and any objects they might contain. The entire graph object is new so that changes in the copy do not affect the original object. (see Python's *copy.deepcopy*)

Data Reference (Shallow) – For a shallow copy the graph structure is copied but the edge, node and graph attribute dicts are references to those in the original graph. This saves time and memory but could cause confusion if you change an attribute in one graph and it changes the attribute in the other. NetworkX does not provide this level of shallow copy.

Independent Shallow – This copy creates new independent attribute dicts and then does a shallow copy of the attributes. That is, any attributes that are containers are shared between the new graph and the original. This is exactly what *dict.copy()* provides. You can obtain this style copy using:

```
>>> G = nx.path_graph(5)
>>> H = G.copy()
>>> H = G.copy(as_view=False)
>>> H = nx.Graph(G)
>>> H = G.__class__(G)
```

Fresh Data – For fresh data, the graph structure is copied while new empty data attribute dicts are created. The resulting graph is independent of the original and it has no edge, node or graph attributes. Fresh copies are not enabled. Instead use:

```
>>> H = G.__class__()
>>> H.add_nodes_from(G)
>>> H.add_edges_from(G.edges)
```

View – Inspired by dict-views, graph-views act like read-only versions of the original graph, providing a copy of the original structure without requiring any memory for copying the information.

See the Python copy module for more information on shallow and deep copies, <https://docs.python.org/3/library/copy.html>.

Parameters *as_view* (*bool*, optional (default=False)) – If True, the returned graph-view provides a read-only view of the original graph without actually copying any data.

Returns *G* – A copy of the graph.

Return type Graph

See also:

to_directed return a directed copy of the graph.

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> H = G.copy()
```

property degree

A DegreeView for the Graph as *G.degree* or *G.degree()*.

The node degree is the number of edges adjacent to the node. The weighted node degree is the sum of the edge weights for edges incident to that node.

This object provides an iterator for (node, degree) as well as lookup for the degree for a single node.

Parameters

- **nbunch** (*single node, container, or all nodes (default= all nodes)*) – The view will only report edges incident to these nodes.
- **weight** (*string or None, optional (default=None)*) – The name of an edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.

Returns

- *If a single node is requested*
- **deg** (*int*) – Degree of the node
- *OR if multiple nodes are requested*
- **nd_view** (*A DegreeView object capable of iterating (node, degree) pairs*)

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.degree[0] # node 0 has degree 1
1
>>> list(G.degree([0, 1, 2]))
[(0, 1), (1, 2), (2, 2)]
```

edge_attr_dict_factory

alias of `dict`

edge_subgraph(*edges*)

Returns the subgraph induced by the specified edges.

The induced subgraph contains each edge in *edges* and each node incident to any one of those edges.

Parameters *edges* (*iterable*) – An iterable of edges in this graph.

Returns *G* – An edge-induced subgraph of this graph with the same edge attributes.

Return type Graph

Notes

The graph, edge, and node attributes in the returned subgraph view are references to the corresponding attributes in the original graph. The view is read-only.

To create a full graph version of the subgraph with its own copy of the edge or node attributes, use:

```
G.edge_subgraph(edges).copy()
```

Examples

```
>>> G = nx.path_graph(5)
>>> H = G.edge_subgraph([(0, 1), (3, 4)])
>>> list(H.nodes)
[0, 1, 3, 4]
>>> list(H.edges)
[(0, 1), (3, 4)]
```

property edges

An EdgeView of the Graph as *G.edges* or *G.edges()*.

`edges(self, nbunch=None, data=False, default=None)`

The EdgeView provides set-like operations on the edge-tuples as well as edge attribute lookup. When called, it also provides an EdgeDataView object which allows control of access to edge attributes (but does not provide set-like operations). Hence, *G.edges[u, v]['color']* provides the value of the color attribute for edge (*u*, *v*) while *for (u, v, c) in G.edges.data('color', default='red')*: iterates through all the edges yielding the color attribute with default 'red' if no color attribute exists.

Parameters

- **nbunch** (*single node, container, or all nodes (default= all nodes)*) – The view will only report edges incident to these nodes.

- **data** (*string or bool, optional (default=False)*) – The edge attribute returned in 3-tuple (u, v, ddict[data]). If True, return edge attribute dict in 3-tuple (u, v, ddict). If False, return 2-tuple (u, v).
- **default** (*value, optional (default=None)*) – Value used for edges that don't have the requested attribute. Only relevant if data is not True or False.

Returns **edges** – A view of edge attributes, usually it iterates over (u, v) or (u, v, d) tuples of edges, but can also be used for attribute lookup as `edges[u, v]['foo']`.

Return type EdgeView

Notes

Nodes in nbunch that are not in the graph will be (quietly) ignored. For directed graphs this returns the out-edges.

Examples

```
>>> G = nx.path_graph(3) # or MultiGraph, etc
>>> G.add_edge(2, 3, weight=5)
>>> [e for e in G.edges]
[(0, 1), (1, 2), (2, 3)]
>>> G.edges.data() # default data is {} (empty dict)
EdgeDataView([(0, 1, {}), (1, 2, {}), (2, 3, {'weight': 5})])
>>> G.edges.data("weight", default=1)
EdgeDataView([(0, 1, 1), (1, 2, 1), (2, 3, 5)])
>>> G.edges([0, 3]) # only edges incident to these nodes
EdgeDataView([(0, 1), (3, 2)])
>>> G.edges(0) # only edges incident to a single node (use G.adj[0]?)
EdgeDataView([(0, 1)])
```

get_edge_data(u, v, default=None)

Returns the attribute dictionary associated with edge (u, v).

This is identical to `G[u][v]` except the default is returned instead of an exception if the edge doesn't exist.

Parameters

- **u** (*nodes*) –
- **v** (*nodes*) –
- **default** (*any Python object (default=None)*) – Value to return if the edge (u, v) is not found.

Returns **edge_dict** – The edge attribute dictionary.

Return type dictionary

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G[0][1]
{}
```

Warning: Assigning to $G[u][v]$ is not permitted. But it is safe to assign attributes $G[u][v]['foo']$

```
>>> G[0][1]["weight"] = 7
>>> G[0][1]["weight"]
7
>>> G[1][0]["weight"]
7
```

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.get_edge_data(0, 1) # default edge data is {}
{}
>>> e = (0, 1)
>>> G.get_edge_data(*e) # tuple form
{}
>>> G.get_edge_data("a", "b", default=0) # edge not in graph, return 0
0
```

`graph_attr_dict_factory`

alias of `dict`

`has_edge(u, v)`

Returns True if the edge (u, v) is in the graph.

This is the same as v in $G[u]$ without `KeyError` exceptions.

Parameters

- **u (nodes)** – Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) Python objects.
- **v (nodes)** – Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) Python objects.

Returns `edge_ind` – True if edge is in the graph, False otherwise.

Return type `bool`

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.has_edge(0, 1) # using two nodes
True
>>> e = (0, 1)
>>> G.has_edge(*e) # e is a 2-tuple (u, v)
True
>>> e = (0, 1, {"weight": 7})
>>> G.has_edge(*e[2:]) # e is a 3-tuple (u, v, data_dictionary)
True
```

The following syntax are equivalent:

```
>>> G.has_edge(0, 1)
True
>>> 1 in G[0] # though this gives KeyError if 0 not in G
True
```

has_node(*n*)

Returns True if the graph contains the node *n*.

Identical to *n in G*

Parameters *n* (*node*) –

Examples

```
>>> G = nx.path_graph(3) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.has_node(0)
True
```

It is more readable and simpler to use

```
>>> 0 in G
True
```

is_directed()

Returns True if graph is directed, False otherwise.

is_multigraph()

Returns True if graph is a multigraph, False otherwise.

property name

String identifier of the graph.

This graph attribute appears in the attribute dict *G.graph* keyed by the string “*name*”. as well as an attribute (technically a property) *G.name*. This is entirely user controlled.

nbunch_iter(*nbunch=None*)

Returns an iterator over nodes contained in *nbunch* that are also in the graph.

The nodes in *nbunch* are checked for membership in the graph and if not are silently ignored.

Parameters *nbunch* (*single node, container, or all nodes (default= all nodes)*) – The view will only report edges incident to these nodes.

Returns *niter* – An iterator over nodes in *nbunch* that are also in the graph. If *nbunch* is None, iterate over all nodes in the graph.

Return type iterator

Raises **NetworkXError** – If *nbunch* is not a node or sequence of nodes. If a node in *nbunch* is not hashable.

See also:

Graph.__iter__

Notes

When `nbunch` is an iterator, the returned iterator yields values directly from `nbunch`, becoming exhausted when `nbunch` is exhausted.

To test whether `nbunch` is a single node, one can use “if `nbunch` in `self`:”, even after processing with this routine.

If `nbunch` is not a node or a (possibly empty) sequence/iterator or `None`, a `NetworkXError` is raised. Also, if any object in `nbunch` is not hashable, a `NetworkXError` is raised.

`neighbors(n)`

Returns an iterator over all neighbors of node `n`.

This is identical to `iter(G[n])`

Parameters `n (node)` – A node in the graph

Returns `neighbors` – An iterator over all neighbors of node `n`

Return type iterator

Raises `NetworkXError` – If the node `n` is not in the graph.

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> [n for n in G.neighbors(0)]
[1]
```

Notes

Alternate ways to access the neighbors are `G.adj[n]` or `G[n]`:

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_edge("a", "b", weight=7)
>>> G["a"]
AtlasView({'b': {'weight': 7}})
>>> G = nx.path_graph(4)
>>> [n for n in G[0]]
[1]
```

`node_attr_dict_factory`

alias of `dict`

`node_dict_factory`

alias of `dict`

`property nodes`

A `NodeView` of the `Graph` as `G.nodes` or `G.nodes()`.

Can be used as `G.nodes` for data lookup and for set-like operations. Can also be used as `G.nodes(data='color', default=None)` to return a `NodeDataView` which reports specific node data but no set operations. It presents a dict-like interface as well with `G.nodes.items()` iterating over *(node, nodedata)* 2-tuples and `G.nodes[3]['foo']` providing the value of the *foo* attribute for node 3. In addition, a view `G.nodes.data('foo')` provides a dict-like interface to the *foo* attribute of each node. `G.nodes.data('foo', default=1)` provides a default for nodes that do not have attribute *foo*.

Parameters

- **data** (*string or bool, optional (default=False)*) – The node attribute returned in 2-tuple (n, ddict[data]). If True, return entire node attribute dict as (n, ddict). If False, return just the nodes n.
- **default** (*value, optional (default=None)*) – Value used for nodes that don't have the requested attribute. Only relevant if data is not True or False.

Returns

Allows set-like operations over the nodes as well as node attribute dict lookup and calling to get a NodeDataView. A NodeDataView iterates over (*n, data*) and has no set operations. A NodeView iterates over *n* and includes set operations.

When called, if data is False, an iterator over nodes. Otherwise an iterator of 2-tuples (node, attribute value) where the attribute is specified in *data*. If data is True then the attribute becomes the entire data dictionary.

Return type NodeView

Notes

If your node data is not needed, it is simpler and equivalent to use the expression `for n in G`, or `list(G)`.

Examples

There are two simple ways of getting a list of all nodes in the graph:

```
>>> G = nx.path_graph(3)
>>> list(G.nodes)
[0, 1, 2]
>>> list(G)
[0, 1, 2]
```

To get the node data along with the nodes:

```
>>> G.add_node(1, time="5pm")
>>> G.nodes[0]["foo"] = "bar"
>>> list(G.nodes(data=True))
[(0, {'foo': 'bar'}), (1, {'time': '5pm'}), (2, {})]
>>> list(G.nodes.data())
[(0, {'foo': 'bar'}), (1, {'time': '5pm'}), (2, {})]
```

```
>>> list(G.nodes(data="foo"))
[(0, 'bar'), (1, None), (2, None)]
>>> list(G.nodes.data("foo"))
[(0, 'bar'), (1, None), (2, None)]
```

```
>>> list(G.nodes(data="time"))
[(0, None), (1, '5pm'), (2, None)]
>>> list(G.nodes.data("time"))
[(0, None), (1, '5pm'), (2, None)]
```

```
>>> list(G.nodes(data="time", default="Not Available"))
[(0, 'Not Available'), (1, '5pm'), (2, 'Not Available')]
>>> list(G.nodes.data("time", default="Not Available"))
[(0, 'Not Available'), (1, '5pm'), (2, 'Not Available')]
```

If some of your nodes have an attribute and the rest are assumed to have a default attribute value you can create a dictionary from node/attribute pairs using the *default* keyword argument to guarantee the value is never None:

```
>>> G = nx.Graph()
>>> G.add_node(0)
>>> G.add_node(1, weight=2)
>>> G.add_node(2, weight=3)
>>> dict(G.nodes(data="weight", default=1))
{0: 1, 1: 2, 2: 3}
```

number_of_edges(*u=None, v=None*)

Returns the number of edges between two nodes.

Parameters

- **u** (*nodes, optional (default=all edges)*) – If *u* and *v* are specified, return the number of edges between *u* and *v*. Otherwise return the total number of all edges.
- **v** (*nodes, optional (default=all edges)*) – If *u* and *v* are specified, return the number of edges between *u* and *v*. Otherwise return the total number of all edges.

Returns **nedges** – The number of edges in the graph. If nodes *u* and *v* are specified return the number of edges between those nodes. If the graph is directed, this only returns the number of edges from *u* to *v*.

Return type `int`

See also:

[*size*](#)

Examples

For undirected graphs, this method counts the total number of edges in the graph:

```
>>> G = nx.path_graph(4)
>>> G.number_of_edges()
3
```

If you specify two nodes, this counts the total number of edges joining the two nodes:

```
>>> G.number_of_edges(0, 1)
1
```

For directed graphs, this method can count the total number of directed edges from *u* to *v*:

```
>>> G = nx.DiGraph()
>>> G.add_edge(0, 1)
>>> G.add_edge(1, 0)
>>> G.number_of_edges(0, 1)
1
```

number_of_nodes()

Returns the number of nodes in the graph.

Returns `nnodes` – The number of nodes in the graph.

Return type `int`

See also:

`order` identical method

`__len__` identical method

Examples

```
>>> G = nx.path_graph(3) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.number_of_nodes()
3
```

order()

Returns the number of nodes in the graph.

Returns `nnodes` – The number of nodes in the graph.

Return type `int`

See also:

`number_of_nodes` identical method

`__len__` identical method

Examples

```
>>> G = nx.path_graph(3) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.order()
3
```

remove_edge(*u*, *v*)

Remove the edge between *u* and *v*.

Parameters

- **u** (*nodes*) – Remove the edge between nodes *u* and *v*.
- **v** (*nodes*) – Remove the edge between nodes *u* and *v*.

Raises **NetworkXError** – If there is not an edge between *u* and *v*.

See also:

`remove_edges_from` remove a collection of edges

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, etc
>>> G.remove_edge(0, 1)
>>> e = (1, 2)
>>> G.remove_edge(*e) # unpacks e from an edge tuple
>>> e = (2, 3, {"weight": 7}) # an edge with attribute data
>>> G.remove_edge(*e[:2]) # select first part of edge tuple
```

remove_edges_from(ebunch)

Remove all edges specified in ebunch.

Parameters **ebunch** (*list or container of edge tuples*) – Each edge given in the list or container will be removed from the graph. The edges can be:

- 2-tuples (u, v) edge between u and v.
- 3-tuples (u, v, k) where k is ignored.

See also:

[*remove_edge*](#) remove a single edge

Notes

Will fail silently if an edge in ebunch is not in the graph.

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> ebunch = [(1, 2), (2, 3)]
>>> G.remove_edges_from(ebunch)
```

remove_node(n)

Remove node n.

Removes the node n and all adjacent edges. Attempting to remove a non-existent node will raise an exception.

Parameters **n** (*node*) – A node in the graph

Raises **NetworkXError** – If n is not in the graph.

See also:

[*remove_nodes_from*](#)

Examples

```
>>> G = nx.path_graph(3) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> list(G.edges)
[(0, 1), (1, 2)]
>>> G.remove_node(1)
>>> list(G.edges)
[]
```

remove_nodes_from(*nodes*)

Remove multiple nodes.

Parameters *nodes* (*iterable container*) – A container of nodes (list, dict, set, etc.). If a node in the container is not in the graph it is silently ignored.

See also:

[*remove_node*](#)

Examples

```
>>> G = nx.path_graph(3) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> e = list(G.nodes)
>>> e
[0, 1, 2]
>>> G.remove_nodes_from(e)
>>> list(G.nodes)
[]
```

size(*weight=None*)

Returns the number of edges or total of all edge weights.

Parameters *weight* (*string or None, optional (default=None)*) – The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1.

Returns

size – The number of edges or (if weight keyword is provided) the total weight sum.

If weight is None, returns an int. Otherwise a float (or more general numeric if the weights are more general).

Return type numeric

See also:

[*number_of_edges*](#)

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.size()
3
```

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_edge("a", "b", weight=2)
>>> G.add_edge("b", "c", weight=4)
>>> G.size()
2
>>> G.size(weight="weight")
6.0
```

subgraph(nodes)

Returns a SubGraph view of the subgraph induced on *nodes*.

The induced subgraph of the graph contains the nodes in *nodes* and the edges between those nodes.

Parameters *nodes* (*list*, *iterable*) – A container of nodes which will be iterated through once.

Returns *G* – A subgraph view of the graph. The graph structure cannot be changed but node/edge attributes can and are shared with the original graph.

Return type SubGraph View

Notes

The graph, edge and node attributes are shared with the original graph. Changes to the graph structure is ruled out by the view, but changes to attributes are reflected in the original graph.

To create a subgraph with its own copy of the edge/node attributes use: `G.subgraph(nodes).copy()`

For an inplace reduction of a graph to a subgraph you can remove nodes: `G.remove_nodes_from([n for n in G if n not in set(nodes)])`

Subgraph views are sometimes NOT what you want. In most cases where you want to do more than simply look at the induced edges, it makes more sense to just create the subgraph as its own graph with code like:

```
# Create a subgraph SG based on a (possibly multigraph) G
SG = G.__class__()
SG.add_nodes_from((n, G.nodes[n]) for n in largest_wcc)
if SG.is_multigraph():
    SG.add_edges_from((n, nbr, key, d)
                      for n, nbrs in G.adj.items() if n in largest_wcc
                      for nbr, keydict in nbrs.items() if nbr in largest_wcc
                      for key, d in keydict.items())
else:
    SG.add_edges_from((n, nbr, d)
                      for n, nbrs in G.adj.items() if n in largest_wcc
                      for nbr, d in nbrs.items() if nbr in largest_wcc)
SG.graph.update(G.graph)
```

Examples

```
>>> G = nx.path_graph(4) # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> H = G.subgraph([0, 1, 2])
>>> list(H.edges)
[(0, 1), (1, 2)]
```

to_directed(*as_view=False*)

Returns a directed representation of the graph.

Returns **G** – A directed graph with the same name, same nodes, and with each edge (u, v, data) replaced by two directed edges (u, v, data) and (v, u, data).

Return type DiGraph

Notes

This returns a “deepcopy” of the edge, node, and graph attributes which attempts to completely copy all of the data and references.

This is in contrast to the similar `D=DiGraph(G)` which returns a shallow copy of the data.

See the Python copy module for more information on shallow and deep copies, <https://docs.python.org/3/library/copy.html>.

Warning: If you have subclassed Graph to use dict-like objects in the data structure, those changes do not transfer to the DiGraph created by this method.

Examples

```
>>> G = nx.Graph() # or MultiGraph, etc
>>> G.add_edge(0, 1)
>>> H = G.to_directed()
>>> list(H.edges)
[(0, 1), (1, 0)]
```

If already directed, return a (deep) copy

```
>>> G = nx.DiGraph() # or MultiDiGraph, etc
>>> G.add_edge(0, 1)
>>> H = G.to_directed()
>>> list(H.edges)
[(0, 1)]
```

to_directed_class()

Returns the class to use for empty directed copies.

If you subclass the base classes, use this to designate what directed class to use for `to_directed()` copies.

to_undirected(*as_view=False*)

Returns an undirected copy of the graph.

Parameters **as_view** (*bool (optional, default=False)*) – If True return a view of the original undirected graph.

Returns **G** – A deepcopy of the graph.

Return type Graph/MultiGraph

See also:

Graph, [copy](#), [add_edge](#), [add_edges_from](#)

Notes

This returns a “deepcopy” of the edge, node, and graph attributes which attempts to completely copy all of the data and references.

This is in contrast to the similar $G = nx.DiGraph(D)$ which returns a shallow copy of the data.

See the Python copy module for more information on shallow and deep copies, <https://docs.python.org/3/library/copy.html>.

Warning: If you have subclassed DiGraph to use dict-like objects in the data structure, those changes do not transfer to the Graph created by this method.

Examples

```
>>> G = nx.path_graph(2) # or MultiGraph, etc
>>> H = G.to_directed()
>>> list(H.edges)
[(0, 1), (1, 0)]
>>> G2 = H.to_undirected()
>>> list(G2.edges)
[(0, 1)]
```

to_undirected_class()

Returns the class to use for empty undirected copies.

If you subclass the base classes, use this to designate what directed class to use for *to_directed()* copies.

update(edges=None, nodes=None)

Update the graph using nodes/edges/graphs as input.

Like dict.update, this method takes a graph as input, adding the graph’s nodes and edges to this graph. It can also take two inputs: edges and nodes. Finally it can take either edges or nodes. To specify only nodes the keyword *nodes* must be used.

The collections of edges and nodes are treated similarly to the *add_edges_from*/*add_nodes_from* methods. When iterated, they should yield 2-tuples (u, v) or 3-tuples (u, v, datadict).

Parameters

- **edges** (*Graph object, collection of edges, or None*) – The first parameter can be a graph or some edges. If it has attributes *nodes* and *edges*, then it is taken to be a Graph-like object and those attributes are used as collections of nodes and edges to be added to the graph. If the first parameter does not have those attributes, it is treated as a collection of edges and added to the graph. If the first argument is None, no edges are added.
- **nodes** (*collection of nodes, or None*) – The second parameter is treated as a collection of nodes to be added to the graph unless it is None. If *edges* is None and *nodes* is None an exception is raised. If the first parameter is a Graph, then *nodes* is ignored.

Examples

```
>>> G = nx.path_graph(5)
>>> G.update(nx.complete_graph(range(4, 10)))
>>> from itertools import combinations
>>> edges = (
...     (u, v, {"power": u * v})
...     for u, v in combinations(range(10, 20), 2)
...     if u * v < 225
... )
>>> nodes = [1000] # for singleton, use a container
>>> G.update(edges, nodes)
```

Notes

If you want to update the graph using an adjacency structure it is straightforward to obtain the edges/nodes from adjacency. The following examples provide common cases, your adjacency may be slightly different and require tweaks of these examples:

```
>>> # dict-of-set/list/tuple
>>> adj = {1: {2, 3}, 2: {1, 3}, 3: {1, 2}}
>>> e = [(u, v) for u, nbrs in adj.items() for v in nbrs]
>>> G.update(edges=e, nodes=adj)
```

```
>>> DG = nx.DiGraph()
>>> # dict-of-dict-of-attribute
>>> adj = {1: {2: 1.3, 3: 0.7}, 2: {1: 1.4}, 3: {1: 0.7}}
>>> e = [
...     (u, v, {"weight": d})
...     for u, nbrs in adj.items()
...     for v, d in nbrs.items()
... ]
>>> DG.update(edges=e, nodes=adj)
```

```
>>> # dict-of-dict-of-dict
>>> adj = {1: {2: {"weight": 1.3}, 3: {"color": 0.7, "weight": 1.2}}}
>>> e = [
...     (u, v, {"weight": d})
...     for u, nbrs in adj.items()
...     for v, d in nbrs.items()
... ]
>>> DG.update(edges=e, nodes=adj)
```

```
>>> # predecessor adjacency (dict-of-set)
>>> pred = {1: {2, 3}, 2: {3}, 3: {3}}
>>> e = [(v, u) for u, nbrs in pred.items() for v in nbrs]
```

```
>>> # MultiGraph dict-of-dict-of-dict-of-attribute
>>> MDG = nx.MultiDiGraph()
>>> adj = {
...     1: {2: {0: {"weight": 1.3}, 1: {"weight": 1.2}}},
```

(continues on next page)

(continued from previous page)

```
...     3: {2: {0: {"weight": 0.7}}},
... }
>>> e = [
...     (u, v, ekey, d)
...     for u, nbrs in adj.items()
...     for v, keydict in nbrs.items()
...     for ekey, d in keydict.items()
... ]
>>> MDG.update(edges=e)
```

See also:

[`add\_edges\_from`](#) add multiple edges to a graph

[`add\_nodes\_from`](#) add multiple nodes to a graph

1.12.6 Schedule Module

<i>Schedule</i>	Class handling any EnergyPlus schedule object.
---------------------------------	--

archetypal.schedule.Schedule

class archetypal.schedule.**Schedule**(*Name: str, start_day_of_the_week: FrozenSet[typing_extensions.Literal[0, 1, 2, 3, 4, 5, 6]] = 0, strict: bool = False, Type: Optional[Union[str, archetypal.schedule.ScheduleTypeLimits]] = None, Values: Optional[Union[List[Union[int, float]], numpy.ndarray]] = None, **kwargs)*

Class handling any EnergyPlus schedule object.

Initialize object.

Parameters

- **Name** (*str*) – The schedule name in the idf model.
- **start_day_of_the_week** (*int*) – 0-based day of week (Monday=0). Default is None which looks for the start day in the IDF model.
- **strict** (*bool*) – if True, schedules that have the Field-Sets such as Holidays and CustomDay will raise an error if they are absent from the IDF file. If False, any missing qualifiers will be ignored.
- **Type** (*str, ScheduleTypeLimits*) – This field contains a reference to the Schedule Type Limits object. If found in a list of Schedule Type Limits (see above), then the restrictions from the referenced object will be used to validate the current field values. If None, no validation will occur.
- **Values** (*ndarray*) – A 24 or 8760 list of schedule values.
- ****kwargs** –

property Type

Get or set the schedule type limits object. Can be None.

property Values

Get or set the list of schedule values.

property Name

Get or set the name of the schedule.

classmethod from_values(Name: *str*, Values: List[Union[int, float]], Type: *str* = 'Fraction', **kwargs)

Create a Schedule from a list of Values.

classmethod from_epbunch(epbunch, strict=False, Type=None, **kwargs)

Create a Schedule from an epbunch.

Parameters

- **epbunch** –
- **strict** –
- ****kwargs** –

classmethod constant_schedule(value=1, Name='AlwaysOn', Type='Fraction', **kwargs)

Initialize a schedule with a constant value for the whole year.

Defaults to a schedule with a value of 1, named 'AlwaysOn'.

Parameters

- **value** (*float*, *optional*) – The value for the constant schedule. Defaults to 1.
- **Name** (*str*, *optional*) – The name of the schedule. Defaults to Always On.
- ****kwargs** –

property all_values: `numpy.ndarray`

Return numpy array of schedule Values.

property max

Get the maximum value of the schedule.

property min

Get the minimum value of the schedule.

property mean

Get the mean value of the schedule.

property series

Return an EnergySeries.

static get_schedule_type_limits_name(epbunch)

Return the Schedule Type Limits name associated to this schedule.

property startDate

Get the start date of the schedule. Satisfies *startDayOfTheWeek*.

scale(diversity=0.1)

Scale the schedule values by a diversity factor around the average.

replace(new_values: *pandas.core.series.Series*)

Replace values with new values while keeping the full load hours constant.

Time steps that are not specified in *new_values* will be adjusted to keep the full load hours of the schedule constant. No check whether the new schedule stays between the bounds set by self.Type is done. Be aware.

plot(**kwargs)

Plot the schedule. Implements the .loc accessor on the series object.

Notes

Plotting can also be achieved through the series property: `Schedule.series.plot()`.

Examples

```
>>> from archetypal import IDF
>>> idf = IDF()
>>> epbunch = idf.schedules_dict["NECB-A-Thermostat Setpoint-Heating"]
>>> s = Schedule.from_epbunch(epbunch)
>>> )
>>> s.plot(drawstyle="steps-post")
```

Parameters ****kwargs** (*dict*) – keyword arguments passed to `EnergySeries.plot()`.

`plot2d(**kwargs)`

Plot the carpet plot of the schedule. Plot rectangular data as a color-encoded 2d-matrix.

Parameters

- **periodlength** (*int*) – The period length to show on the yaxis. By default, the period length will be calculated to represent one day of data on the yaxis and days on the xaxis.
- **vmin** (*float*) – The data value that defines 0.0 in the normalization. Defaults to the min value of the dataset.
- **vmax** (*float*) – The data value that defines 1.0 in the normalization. Defaults to the the max value of the dataset.
- **vcenter** (*float*) – The data value that defines 0.5 in the normalization.
- **axis_off** (*bool*) – If True, no axis is plotted.
- **cmap** (*str* or *Colormap*) – Colormap to select colors from. If string, load colormap with that name from matplotlib.
- **figsize** (*tuple*) – Size of a figure object. A tuple (width, height) in inches.
- **show** (*bool*) – whether to display the figure or not.
- **save** (*bool*) – whether to save the figure to disk or not.
- **close** (*bool*) – close the figure (only if show equals False) to prevent display.
- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg').
- **colorbar** (*bool*) – If True, plot the colorbar.
- **ax** (*Axes*) – An axes of the current figure.
- **filename** (*string*) – the name of the file to save.
- **extent** (*str* or *Bbox*) – Bounding box in inches: only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- **ylabel** (*str*) – Set the label for the y-axis.
- **xlabel** (*str*) – Set the label for the x-axis.
- **ax_title** (*bool* or *str*) – Title to use for the ax. If True, the Series name is used.

- ****kwargs** – Options to pass to matplotlib imshow method.

Returns

tuple containing:

fig: is the Matplotlib Figure object ax: a single axis object.

Return type (tuple)

to_year_week_day()

Convert to three-tuple epbunch given an idf model.

Returns ‘Schedule:Year’, ‘Schedule:Week:Daily’ and ‘Schedule:Day:Hourly’ representations.

Returns

3-element tuple containing

- **yearly** (*Schedule*): The yearly schedule object
- **weekly** (*list of Schedule*): The list of weekly schedule objects
- **daily** (*list of Schedule*): The list of daily schedule objects

combine(*other*, *weights=None*, *quantity=None*)

Combine two schedule objects together.

Parameters

- **other** (*Schedule*) – the other Schedule object to combine with.
- **weights** (*list-like, optional*) – A list-like object of len 2. If None, equal weights are used.
- **quantity** – scalar value that will be multiplied by self before the averaging occurs. This ensures that the resulting schedule returns the correct integrated value.

Returns the combined Schedule object.

Return type (*Schedule*)

1.12.7 Data Portal

<i>tabula_available_buildings</i>	Returns all available building types for a specific country.
<i>tabula_building_details_sheet</i>	How to format <code>code_building</code> .
<i>tabula_system</i>	Return system level information from TABULA archetypes.
<i>tabula_system_request</i>	Returns:
<i>openei_api_request</i>	Query the OpenEI.org API.
<i>nrel_api_cbr_request</i>	Query the NREL Commercial Building Resource Database
<i>nrel_bcl_api_request</i>	Send a request to the Building Component Library API via HTTP GET and return the JSON response.
<i>stat_can_request</i>	Send a request to the StatCan API via HTTP GET and return the JSON response.

continues on next page

Table 1.7 – continued from previous page

<i>stat_can_geo_request</i>	param type "json" or "xml". json = json response format and xml = xml
<i>download_bld_window</i>	Find window constructions corresponding to a combination of a u_factor, shgc and visible transmittance and download their idf file to disk.

archetypal.dataportal.tabula_available_buildings

```
archetypal.dataportal.tabula_available_buildings(country_name='France')
```

Returns all available building types for a specific country.

Parameters `country_name` (*str*) – The name of the country. `pycountry` is used to resolve country names. Therefore, a country code (e.g. “FRA”) can be passed as well.

archetypal.dataportal.tabula_building_details_sheet

[illegible]

How to format `code_building`. Format the `code_building` string as such:

Parameters

- **code_building** (*str*) – The building code string.

Whole building code e.g.:

AT.MT.AB.02.Gen.ReEx.001.001"

```

| | | | | | |__code_variantnumber
| | | | | | |__code_num
| | | | | | |__code_type
| | | | | | |__code_additional_parameter
| | | | | | |__code_construcionyearclass
| | | | | | |__code_buildingsizeclass
| | | | | | |__code_typologyregion
| | | | | | |__code_country

```

- **code_country** (*str*) – Country name or International Country Code (ISO 3166-1-alpha-2 code). Input as ‘France’ will work equally as ‘FR’.
- **code_typologyregion** (*str*) – N for national; otherwise specific codes representing regions in a given country
- **code_buildingsizeclass** (*str*) – 4 standardized classes: ‘SFH’:
- **code_constructionyearclass** (*int* or *str*) – allocation of time bands to classes. Defined nationally (according to significant changes in construction technologies, building codes or available statistical data)

- **code_additional_parameter** (*str*) – 1 unique category. Defines the generic (or basic) typology matrix so that each residential building of a given country can be assigned to one generic type. A further segmentation in subtypes is possible and can be indicated by a specific code. Whereas the generic types must comprise the whole building stock the total of subtypes must be comprehensive. e.g. ‘HR’ (highrises), ‘TFrame’ (timber frame), ‘Semi’ (semi-detached)
- **code_type** – “ReEx” is a code for “real example” and “SyAv” for “Synthetical Average”
- **code_num** – TODO: What is this parameter?
- **code_variantnumber** – the energy performance level 1, 2 and 3. 1: minimum requirements, 2: improved and 3: ambitious or NZEB standard (assumed or announced level of Nearly Zero-Energy Buildings)

Returns The DataFrame from the

Return type `pandas.DataFrame`

archetypal.dataportal.tabula_system

`archetypal.dataportal.tabula_system(code_country, code_boundarycond='SUH', code_variantnumber=1)`
Return system level information from TABULA archetypes.

Parameters

- **code_country** (*str*) – the alpha-2 code of the country. eg. “FR”
- **code_boundarycond** (*str*) – choices are “SUH” and “MUH”.
- **code_variantnumber** (*int*) –

archetypal.dataportal.tabula_system_request

`archetypal.dataportal.tabula_system_request(data)`
Returns:

Examples

`'http://webtool.building-typology.eu/data/matrix/system/detail/IT.SUH.01/dc/1546889637169'`

Parameters `data` (*dict*) – prepared data for html query

archetypal.dataportal.openei_api_request

`archetypal.dataportal.openei_api_request(data)`
Query the OpenEI.org API.

Parameters `data` (*dict* or *OrderedDict*) – key-value pairs of parameters to post to the API.

Returns the json response

Return type `dict`

archetypal.dataportal.nrel_api_cbr_request

archetypal.dataportal.nrel_api_cbr_request(*data*)
Query the NREL Commercial Building Resource Database

Examples

```
>>> from archetypal import dataportal
>>> dataportal.nrel_api_cbr_request({'s': 'Commercial'
>>> 'Reference', 'api_key': 'oGZdX1nhars1cTJYTm7M9T12T1Z0vikX9pH0Zudq'})
```

Parameters *data* – a dict of

Returns the json response

Return type dict

Hint: For a detailed description of data arguments, visit [Commercial Building Resource API](#)

archetypal.dataportal.nrel_bcl_api_request

archetypal.dataportal.nrel_bcl_api_request(*data*)
Send a request to the Building Component Library API via HTTP GET and return the JSON response.

Parameters *data* (*dict* or *OrderedDict*) – key-value pairs of parameters to post to the API

Returns the json response

Return type dict

archetypal.dataportal.stat_can_request

archetypal.dataportal.stat_can_request(*type*, *lang*='E', *dguid*='2016A000011124', *topic*=0, *notes*=0, *stat*=0)

Send a request to the StatCan API via HTTP GET and return the JSON response.

Parameters

- **type** (*str*) – “json” or “xml”. json = json response format and xml = xml response format.
- **lang** (*str*) – “E” or “F”. E = English and F = French.
- **dguid** (*str*) – Dissemination Geography Unique Identifier - DGUID. It is an alphanumeric code, composed of four components. It varies from 10 to 21 characters in length. The first 9 characters are fixed in composition and length. Vintage (4) + Type (1) + Schema (4) + Geographic Unique Identifier (1-12). To find dguid, use any GEO_UID (i.e., DGUID) returned by the 2016 Census geography web data service. For more information on the DGUID definition and structure, please refer to the [Dissemination Geography Unique Identifier, Definition and Structure](#) , Statistics Canada catalogue no. 92F0138M-2019001.
- **topic** (*str*) – Integer 0-14 (default=0) where: 1. All topics 2. Aboriginal peoples 3. Education 4. Ethnic origin 5. Families, households and marital status 6. Housing 7. Immigration and citizenship 8. Income 9. Journey to work 10. Labour 11. Language 12. Language of work 13. Mobility 14. Population 15. Visible minority.

- **notes** (*int*) – 0 or 1. 0 = do not include footnotes. 1 = include footnotes.
- **stat** (*int*) – 0 or 1. 0 = counts. 1 = rates.

archetypal.dataportal.stat_can_geo_request

`archetypal.dataportal.stat_can_geo_request(type='json', lang='E', geos='PR', cpt='00')`

Parameters

- **type** (*str*) – “json” or “xml”. json = json response format and xml = xml response format.
- **lang** (*str*) – “E” or “F”. where: E = English F = French.
- **geos** (*str*) – one geographic level code (default = PR). where: CD = Census divisions CMACA = Census metropolitan areas and census agglomerations CSD = Census subdivisions (municipalities) CT = Census tracts DA = Dissemination areas DPL = Designated places ER = Economic regions FED = Federal electoral districts (2013 Representation Order) FSA = Forward sortation areas HR = Health regions (including LHINs and PHUs) POPC-NTR = Population centres PR = Canada, provinces and territories.
- **cpt** (*str*) – one province or territory code (default = 00). where: 00 = All provinces and territories 10 = Newfoundland and Labrador 11 = Prince Edward Island 12 = Nova Scotia 13 = New Brunswick 24 = Quebec 35 = Ontario 46 = Manitoba 47 = Saskatchewan 48 = Alberta 59 = British Columbia 60 = Yukon 61 = Northwest Territories 62 = Nunavut.

archetypal.dataportal.download_bld_window

`archetypal.dataportal.download_bld_window(u_factor, shgc, vis_trans, oauth_key, tolerance=0.05, extension='idf', output_folder=None)`

Find window constructions corresponding to a combination of a `u_factor`, `shgc` and visible transmittance and download their idf file to disk. it is necessary to have an authentication key (see Info below).

Parameters

- **u_factor** (*float or tuple*) – The center of glass u-factor. Pass a range of values by passing a tuple (from, to). If a tuple is passed, *tolerance* is ignored.
- **shgc** (*float or tuple*) – The Solar Heat Gain Coefficient. Pass a range of values by passing a tuple (from, to). If a tuple is passed, *tolerance* is ignored.
- **vis_trans** (*float or tuple*) – The Visible Transmittance. Pass a range of values by passing a tuple (from, to). If a tuple is passed, *tolerance* is ignored.
- **oauth_key** (*str*) – the `Building_Component_Library` authentication key.
- **tolerance** (*float*) – relative tolerance for the input values. Default is 0.05 (5%).
- **extension** (*str*) – specify the extension of the file to download. (default: ‘idf’)
- **output_folder** (*str, optional*) – specify folder to save response data to. Defaults to `settings.data_folder`.

Returns

a list of IDF files containing window objects matching the parameters.

Return type (list of `archetypal.IDF`)

Note: An authentication key from NREL is required to download building components. Register at [Building_Component_Library](#)

1.12.8 EnergyDataFrame

Note: EnergyDataFrame is now part of its own package [energy-pandas](#).

1.12.9 EnergySeries

Note: EnergySeries is now part of its own package [energy-pandas](#).

1.12.10 Report Data

ReportData.__init__

<i>ReportData.from_sql_dict</i>	Create from dictionary.
<i>ReportData.from_sqlite</i>	Read an EnergyPlus eplusout.sql file.
<i>ReportData.filter_report_data</i>	filter ReportData using specific keywords.

archetypal.reportdata.ReportData.__init__

`ReportData.__init__(data=None, index: Axes | None = None, columns: Axes | None = None, dtype: Dtype | None = None, copy: bool | None = None)`

archetypal.reportdata.ReportData.from_sql_dict

classmethod `ReportData.from_sql_dict(sql_dict)`
Create from dictionary.

archetypal.reportdata.ReportData.from_sqlite

classmethod `ReportData.from_sqlite(sqlite_file, table_name, warmup_flag=0, environment_type=3, reporting_frequency=None)`
Read an EnergyPlus eplusout.sql file.

Parameters

- **sqlite_file** (*str*) – The path of the sqlite3 file.
- **table_name** (*str*, *optional*) – Filter results by a specific table name.
- **warmup_flag** (*int*) – 1 during warmup, 0 otherwise. Defaults to 0.

- **environment_type** (*int*) – An enumeration of the environment type. (1 = Design Day, 2 = Design Run Period, 3 = Weather Run Period) See the various SizingPeriod objects and the RunPeriod object for details.
- **reporting_frequency** (*str*, *optional*) – “HVAC System Timestep”, “Zone Timestep”, “Hourly”, “Daily”, “Monthly”, “Run Period”.

Examples

```
>>> ReportData.from_sqlite("eplusout.sql",
>>>     table_name="Air System Total Heating Energy",
>>>     warmup_flag=0,
>>>     environment_type=1,
>>> )
```

Returns a ReportData which is a subclass of DataFrame.

Return type ReportData

archetypal.reportdata.ReportData.filter_report_data

ReportData.**filter_report_data**(*archetype=None*, *reportdataindex=None*, *timeindex=None*, *reportdatadictionaryindex=None*, *value=None*, *ismeter=None*, *type=None*, *indexgroup=None*, *timesteptype=None*, *keyvalue=None*, *name=None*, *reportingfrequency=None*, *schedulename=None*, *units=None*, *inplace=False*)

filter ReportData using specific keywords. Each keywords can be a tuple of strings (str1, str2, str3) which will return the logical_or on the specific column.

Parameters

- **archetype** (*str* or *tuple*) –
- **reportdataindex** (*str* or *tuple*) –
- **timeindex** (*str* or *tuple*) –
- **reportdatadictionaryindex** (*str* or *tuple*) –
- **value** (*str* or *tuple*) –
- **ismeter** (*str* or *tuple*) –
- **type** (*str* or *tuple*) –
- **indexgroup** (*str* or *tuple*) –
- **timesteptype** (*str* or *tuple*) –
- **keyvalue** (*str* or *tuple*) –
- **name** (*str* or *tuple*) –
- **reportingfrequency** (*str* or *tuple*) –
- **schedulename** (*str* or *tuple*) –
- **units** (*str* or *tuple*) –
- **inplace** (*str* or *tuple*) –

Returns pandas.DataFrame

1.12.11 Tabular Data

<code>TabularData.from_sql</code>	Returns a DataFrame from the ‘TabularDataWithStrings’ table
<code>TabularData.filter_tabular_data</code>	filter RaportData using specific keywords.

archetypal.tabulardata.TabularData.from_sql

classmethod `TabularData.from_sql(sql_dict)`

Returns a DataFrame from the ‘TabularDataWithStrings’ table

Parameters `sql_dict` –

archetypal.tabulardata.TabularData.filter_tabular_data

`TabularData.filter_tabular_data(archetype=None, tabulardataindex=None, value=None, reportname=None, reportforstring=None, tablename=None, rowname=None, columnname=None, units=None, inplace=False)`

filter RaportData using specific keywords. Each keywords can be a tuple of strings (str1, str2, str3) which will return the logical_or on the specific column.

Parameters

- **archetype** (*str* or *tuple*) –
- **tabulardataindex** –
- **value** (*str* or *tuple*) –
- **reportname** –
- **reportforstring** –
- **tablename** –
- **rowname** –
- **columnname** –
- **units** (*str* or *tuple*) –
- **inplace** (*str* or *tuple*) –

Returns pandas.DataFrame

1.12.12 Utils

<code>config</code>	Package configurations.
<code>log</code>	Write a message to the log file and/or print to the the console.
<code>weighted_mean</code>	Compute the weighted average while ignoring NaNs.
<code>top</code>	Compute the highest ranked value weighted by some other variable.
<code>copy_file</code>	Handles a copy of test idf files
<code>load_umi_template</code>	Load umi template file to list of dict.
<code>check_unique_name</code>	Making sure new_name does not already exist
<code>angle</code>	Calculate the angle between 2 vectors
<code>float_round</code>	Makes sure a variable is a float and round it at “n” decimals
<code>timeit</code>	Use this method as a decorator on a function to calculate the time it take to complete.
<code>lcm</code>	This function takes two integers and returns the least common multiple.
<code>recursive_len</code>	Calculate the number of elements in nested list
<code>rotate</code>	Shift list elements to the left
<code>parallel_process</code>	A parallel version of the map function with a progress b

archetypal.utils.config

```
archetypal.utils.config(data_folder=Path('data'), logs_folder=Path('logs'), imgs_folder=Path('images'),
                        cache_folder=Path('cache'), use_cache=False, log_file=False, log_console=False,
                        log_level=20, log_name='archetypal', log_filename='archetypal',
                        useful_idf_objects=['WINDOWMATERIAL:GAS', 'WINDOWMATERIAL:GLAZING',
                        'WINDOWMATERIAL:SIMPLEGLAZINGSYSTEM', 'MATERIAL',
                        'MATERIAL:NOMASS', 'CONSTRUCTION', 'BUILDINGSURFACE:DETAILED',
                        'FENESTRATIONSURFACE:DETAILED', 'SCHEDULE:DAY:INTERVAL',
                        'SCHEDULE:WEEK:DAILY', 'SCHEDULE:YEAR'],
                        umitemplate=Path('data/BostonTemplateLibrary.json'), default_weight_factor='area',
                        ep_version='9-2-0', debug=False)
```

Package configurations. Call this method at the beginning of script or at the top of an interactive python environment to set package-wide settings.

Parameters

- **data_folder** (*str*) – where to save and load data files.
- **logs_folder** (*str*) – where to write the log files.
- **imgs_folder** (*str*) – where to save figures.
- **cache_folder** (*str*) – where to save the simulation results.
- **use_cache** (*bool*) – if True, use a local cache to save/retrieve DataPortal API calls for the same requests.
- **log_file** (*bool*) – if true, save log output to a log file in logs_folder.
- **log_console** (*bool*) – if true, print log output to the console.
- **log_level** (*int*) – one of the logger.level constants.

- **log_name** (*str*) – name of the logger.
- **log_filename** (*str*) – name of the log file.
- **useful_idf_objects** (*list*) – a list of useful idf objects.
- **umitemplate** (*str*) – where the umitemplate is located.
- **default_weight_factor** –
- **ep_version** (*str*) – EnergyPlus version to use. eg. “9-2-0”.
- **debug** (*bool*) – Use debug behavior in various part of code base.

Returns None

archetypal.utils.log

`archetypal.utils.log(message, level=None, name=None, filename=None, avoid_console=False, log_dir=None, verbose=False)`

Write a message to the log file and/or print to the the console.

Parameters

- **message** (*str*) – the content of the message to log
- **level** (*int*) – one of the logger.level constants
- **name** (*str*) – name of the logger
- **filename** (*str*) – name of the log file
- **avoid_console** (*bool*) – If True, don’t print to console for this message only
- **log_dir** (*str*, *optional*) – directory of log file. Defaults to settings.log_folder
- **verbose** – If True, settings.log_console is overridden.

archetypal.utils.weighted_mean

`archetypal.utils.weighted_mean(series, df, weighting_variable)`

Compute the weighted average while ignoring NaNs. Implements `numpy.average()`.

Parameters

- **series** (*pandas.Series*) – the *series* on which to compute the mean.
- **df** (*pandas.DataFrame*) – the *df* containing weighting variables.
- **weighting_variable** (*str or list or tuple*) – Name of weights to use in *df*. If multiple values given, the values are multiplied together.

Returns the weighted average

Return type `numpy.ndarray`

archetypal.utils.top

`archetypal.utils.top(series, df, weighting_variable)`

Compute the highest ranked value weighted by some other variable. Implements

`pandas.DataFrame.nlargest()`.

Parameters

- **series** (*pandas.Series*) – the *series* on which to compute the ranking.
- **df** (*pandas.DataFrame*) – the *df* containing weighting variables.
- **weighting_variable** (*str or list or tuple*) – Name of weights to use in *df*. If multiple values given, the values are multiplied together.

Returns the weighted top ranked variable

Return type `numpy.ndarray`

archetypal.utils.copy_file

`archetypal.utils.copy_file(files, where=None)`

Handles a copy of test idf files

Parameters

- **files** (*str or list*) – path(s) of the file(s) to copy
- **where** (*str*) – path where to save the copy(ies)

archetypal.utils.load_umi_template

`archetypal.utils.load_umi_template(json_template)`

Load umi template file to list of dict.

Parameters `json_template` (*str*) – filepath to an umi json_template.

Returns list of dict.

Return type `list`

archetypal.utils.check_unique_name

`archetypal.utils.check_unique_name(first_letters, count, name, unique_list, suffix=False)`

Making sure new_name does not already exist

Parameters

- **first_letters** (*str*) – string at the beginning of the name, giving a hint on what the variable is.
- **count** (*int*) – increment to create a unique id in the name.
- **name** (*str*) – name that was just created. To be verified that it is unique in this function.
- **unique_list** (*list*) – list where unique names are stored.
- **suffix** (*bool*) –

Returns name that is unique

Return type `new_name (str)`

archetypal.utils.angle

`archetypal.utils.angle(v1, v2, acute=True)`

Calculate the angle between 2 vectors

Parameters

- **v1** (*Vector3D*) – vector 1
- **v2** (*Vector3D*) – vector 2
- **acute** (*bool*) – If True, give the acute angle, else gives the obtuse one.

Returns angle between the 2 vectors in degree

Return type `angle (float)`

archetypal.utils.float_round

`archetypal.utils.float_round(num, n)`

Makes sure a variable is a float and round it at “n” decimals

Parameters

- **num** (*str*, *int*, *float*) – number we want to make sure is a float
- **n** (*int*) – number of decimals

Returns a float rounded number

Return type `num (float)`

archetypal.utils.timeit

`archetypal.utils.timeit(method)`

Use this method as a decorator on a function to calculate the time it take to complete. Uses the `log()` method.

Examples

```
>>> @timeit
>>> def myfunc():
>>>     return 'is a function'
```

Parameters **method** (*function*) – A function.

archetypal.utils.lcm

`archetypal.utils.lcm(x, y)`

This function takes two integers and returns the least common multiple.

archetypal.utils.recursive_len

`archetypal.utils.recursive_len(item)`

Calculate the number of elements in nested list

Parameters `item` (*list*) – list of lists (i.e. nested list)

Returns Total number of elements in nested list

archetypal.utils.rotate

`archetypal.utils.rotate(l, n)`

Shift list elements to the left

Parameters

- `l` (*list*) – list to rotate
- `n` (*int*) – number to shift list to the left

Returns shifted list.

Return type *list*

archetypal.utils.parallel_process

`archetypal.utils.parallel_process(in_dict, function, processors=-1, use_kwargs=True, show_progress=True, position=0, debug=False, executor=None)`

A parallel version of the map function with a progress b

Examples

```
>>> from archetypal import IDF
>>> files = ['tests/input_data/problematic/nat_ventilation_SAMPLE0.idf',
>>>           'tests/input_data/regular/5ZoneNightVent1.idf']
>>> wf = 'tests/input_data/CAN_PQ_Montreal.Intl.AP.716270_CWEC.epw'
>>> rundict = {'file': dict(idfname=file, epw=wf,
>>>                          as_version="9-2-0", annual=True,
>>>                          prep_outputs=True, expandobjects=True,
>>>                          verbose='q')}
>>> for file in files:
>>> result = parallel_process(rundict, IDF, use_kwargs=True)
```

Parameters

- `in_dict` (*dict*) – A dictionary to iterate over. *function* is applied to value and key is used as an identifier.
- `function` (*callable*) – A python function to apply to the elements of `in_dict`

- **processors** (*int*) – The number of cores to use.
- **use_kwargs** (*bool*) – If True, pass the kwargs as arguments to *function*.
- **debug** (*bool*) – If True, will raise any error on any process.
- **position** – Specify the line offset to print the tqdm bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **executor** (*Executor*) –

Returns [function(array[0]), function(array[1]), ...]

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [ref] Tsilingiris, P. T. (2004). On the thermal time constant of structural walls. *Applied Thermal Engineering*, 24(5–6), 743–757. <https://doi.org/10.1016/j.applthermaleng.2003.10.015>

Symbols

`__init__()` (archetypal.reportdata.ReportData method), 190

A

`add_basics()` (archetypal.idfclass.Outputs method), 51
`add_block()` (archetypal.idfclass.IDF method), 38
`add_custom()` (archetypal.idfclass.Outputs method), 51
`add_edge()` (archetypal.zone_graph.ZoneGraph method), 161
`add_edges_from()` (archetypal.zone_graph.ZoneGraph method), 162
`add_idf_object_from_idf_string()` (archetypal.idfclass.IDF method), 46
`add_meter_variables()` (archetypal.idfclass.Outputs method), 51
`add_node()` (archetypal.zone_graph.ZoneGraph method), 163
`add_nodes_from()` (archetypal.zone_graph.ZoneGraph method), 163
`add_output_control()` (archetypal.idfclass.Outputs method), 52
`add_profile_gas_elect_ouputs()` (archetypal.idfclass.Outputs method), 52
`add_schedules()` (archetypal.idfclass.Outputs method), 51
`add_shading_block()` (archetypal.idfclass.IDF method), 39
`add_sql()` (archetypal.idfclass.Outputs method), 52
`add_summary_report()` (archetypal.idfclass.Outputs method), 51
`add_umi_ouputs()` (archetypal.idfclass.Outputs method), 52
`add_umi_template_outputs()` (archetypal.idfclass.Outputs method), 52
`add_weighted_edges_from()` (archetypal.zone_graph.ZoneGraph method), 164
`add_zone()` (archetypal.idfclass.IDF method), 39
`addidfobject()` (archetypal.idfclass.IDF method), 48
`adj` (archetypal.zone_graph.ZoneGraph property), 165
`adjacency()` (archetypal.zone_graph.ZoneGraph method), 165

`adjlist_inner_dict_factory` (archetypal.zone_graph.ZoneGraph attribute), 165
`adjlist_outer_dict_factory` (archetypal.zone_graph.ZoneGraph attribute), 165
`Afn` (archetypal.template.VentilationSetting property), 107
`AfnDischargeC` (archetypal.template.WindowSetting property), 119
`AfnTempSetpoint` (archetypal.template.WindowSetting property), 119
`AfnWindowAvailability` (archetypal.template.WindowSetting property), 119
`all_values` (archetypal.schedule.Schedule property), 183
`all_values` (archetypal.template.schedule.DaySchedule property), 142
`all_values` (archetypal.template.schedule.WeekSchedule property), 148
`all_values` (archetypal.template.schedule.YearSchedule property), 157
`all_values` (archetypal.template.UmiSchedule property), 98
`allow_duplicates` (archetypal.template.BuildingTemplate property), 59
`allow_duplicates` (archetypal.template.constructions.base_construction.ConstructionBase property), 137
`allow_duplicates` (archetypal.template.constructions.base_construction.LayeredConstruction property), 138
`allow_duplicates` (archetypal.template.DomesticHotWaterSetting property), 70
`allow_duplicates` (archetypal.template.GasMaterial property), 74
`allow_duplicates` (archetypal.template.GlazingMaterial property), 78
`allow_duplicates` (archetypal.template.materials.material_base.MaterialBase property), 133

[allow_duplicates](#) (*archetypal.template.OpaqueConstruction* property), 89
[allow_duplicates](#) (*archetypal.template.OpaqueMaterial* property), 94
[allow_duplicates](#) (*archetypal.template.schedule.DaySchedule* property), 144
[allow_duplicates](#) (*archetypal.template.schedule.WeekSchedule* property), 149
[allow_duplicates](#) (*archetypal.template.schedule.YearSchedule* property), 153
[allow_duplicates](#) (*archetypal.template.StructureInformation* property), 103
[allow_duplicates](#) (*archetypal.template.umi_base.UmiBase* property), 131
[allow_duplicates](#) (*archetypal.template.UmiSchedule* property), 98
[allow_duplicates](#) (*archetypal.template.VentilationSetting* property), 106
[allow_duplicates](#) (*archetypal.template.WindowConstruction* property), 115
[allow_duplicates](#) (*archetypal.template.WindowSetting* property), 119
[allow_duplicates](#) (*archetypal.template.ZoneConditioning* property), 66
[allow_duplicates](#) (*archetypal.template.ZoneConstructionSet* property), 128
[allow_duplicates](#) (*archetypal.template.ZoneDefinition* property), 124
[allow_duplicates](#) (*archetypal.template.ZoneLoad* property), 82
[angle\(\)](#) (in module *archetypal.utils*), 196
[anidfobject\(\)](#) (*archetypal.idfclass.IDF* method), 49
[annual](#) (*archetypal.idfclass.IDF* property), 36
[apply\(\)](#) (*archetypal.idfclass.Outputs* method), 52
[area](#) (*archetypal.template.DomesticHotWaterSetting* property), 70
[area](#) (*archetypal.template.VentilationSetting* property), 107
[area](#) (*archetypal.template.WindowSetting* property), 119
[area](#) (*archetypal.template.ZoneConditioning* property), 65
[area](#) (*archetypal.template.ZoneConstructionSet* property), 128
[area](#) (*archetypal.template.ZoneDefinition* property), 125
[area](#) (*archetypal.template.ZoneLoad* property), 83
[as_version](#) (*archetypal.idfclass.IDF* property), 36
[AssemblyCarbon](#) (*archetypal.template.constructions.base_construction.ConstructionBase* property), 136
[AssemblyCarbon](#) (*archetypal.template.constructions.base_construction.LayeredConstruction* property), 138
[AssemblyCarbon](#) (*archetypal.template.OpaqueConstruction* property), 89
[AssemblyCarbon](#) (*archetypal.template.StructureInformation* property), 102
[AssemblyCarbon](#) (*archetypal.template.WindowConstruction* property), 114
[AssemblyCost](#) (*archetypal.template.constructions.base_construction.ConstructionBase* property), 136
[AssemblyCost](#) (*archetypal.template.constructions.base_construction.LayeredConstruction* property), 138
[AssemblyCost](#) (*archetypal.template.OpaqueConstruction* property), 89
[AssemblyCost](#) (*archetypal.template.StructureInformation* property), 102
[AssemblyCost](#) (*archetypal.template.WindowConstruction* property), 114
[AssemblyEnergy](#) (*archetypal.template.constructions.base_construction.ConstructionBase* property), 136
[AssemblyEnergy](#) (*archetypal.template.constructions.base_construction.LayeredConstruction* property), 138
[AssemblyEnergy](#) (*archetypal.template.OpaqueConstruction* property), 89
[AssemblyEnergy](#) (*archetypal.template.StructureInformation* property), 102
[AssemblyEnergy](#) (*archetypal.template.WindowConstruction* property), 114
[AuthorEmails](#) (*archetypal.template.BuildingTemplate* property), 60
[Authors](#) (*archetypal.template.BuildingTemplate* property), 60

B

`block` (`archetypal.idfclass.IDF` property), 35
`bounding_box()` (`archetypal.idfclass.IDF` method), 39
`build_graph()` (`archetypal.umi_template.UmiTemplateLibrary` method), 57

`BuildingTemplate` (class in `archetypal.template`), 57

C

`Category` (`archetypal.template.BuildingTemplate` property), 59

`Category` (`archetypal.template.constructions.base_construction.ConstructionBase` property), 136

`Category` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 138

`Category` (`archetypal.template.DomesticHotWaterSetting` property), 70

`Category` (`archetypal.template.GasMaterial` property), 74

`Category` (`archetypal.template.GlazingMaterial` property), 77

`Category` (`archetypal.template.materials.material_base.MaterialBase` property), 133

`Category` (`archetypal.template.OpaqueConstruction` property), 89

`Category` (`archetypal.template.OpaqueMaterial` property), 93

`Category` (`archetypal.template.schedule.DaySchedule` property), 144

`Category` (`archetypal.template.schedule.WeekSchedule` property), 148

`Category` (`archetypal.template.schedule.YearSchedule` property), 153

`Category` (`archetypal.template.StructureInformation` property), 102

`Category` (`archetypal.template.umi_base.UmiBase` property), 131

`Category` (`archetypal.template.UmiSchedule` property), 98

`Category` (`archetypal.template.VentilationSetting` property), 106

`Category` (`archetypal.template.WindowConstruction` property), 111

`Category` (`archetypal.template.WindowSetting` property), 119

`Category` (`archetypal.template.ZoneConditioning` property), 65

`Category` (`archetypal.template.ZoneConstructionSet` property), 128

`Category` (`archetypal.template.ZoneDefinition` property), 124

`Category` (`archetypal.template.ZoneLoad` property), 82

`centroid` (`archetypal.idfclass.IDF` property), 39

`check_unique_name()` (in module `archetypal.utils`), 195

`clear()` (`archetypal.zone_graph.ZoneGraph` method), 165

`clear_edges()` (`archetypal.zone_graph.ZoneGraph` method), 166

`ClimateZone` (`archetypal.template.BuildingTemplate` property), 60

`combine()` (`archetypal.schedule.Schedule` method), 185

`combine()` (`archetypal.template.DomesticHotWaterSetting` method), 70

`combine()` (`archetypal.template.GlazingMaterial` method), 79

`combine()` (`archetypal.template.OpaqueConstruction` method), 87

`combine()` (`archetypal.template.OpaqueMaterial` method), 95

`combine()` (`archetypal.template.schedule.DaySchedule` method), 144

`combine()` (`archetypal.template.schedule.WeekSchedule` method), 149

`combine()` (`archetypal.template.schedule.YearSchedule` method), 153

`combine()` (`archetypal.template.UmiSchedule` method), 97

`combine()` (`archetypal.template.VentilationSetting` method), 108

`combine()` (`archetypal.template.WindowConstruction` method), 113

`combine()` (`archetypal.template.WindowSetting` method), 121

`combine()` (`archetypal.template.ZoneConditioning` method), 68

`combine()` (`archetypal.template.ZoneConstructionSet` method), 130

`combine()` (`archetypal.template.ZoneDefinition` method), 126

`combine()` (`archetypal.template.ZoneLoad` method), 84

`Comments` (`archetypal.template.BuildingTemplate` property), 59

`Comments` (`archetypal.template.constructions.base_construction.ConstructionBase` property), 136

`Comments` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 138

`Comments` (`archetypal.template.DomesticHotWaterSetting` property), 70

`Comments` (`archetypal.template.GasMaterial` property), 74

`Comments` (`archetypal.template.GlazingMaterial` property), 77

`Comments` (`archetypal.template.materials.material_base.MaterialBase` property), 133

`Comments` (`archetypal.template.OpaqueConstruction` property), 89

`Comments` (`archetypal.template.OpaqueMaterial` property), 93

[Comments](#) ([archetypal.template.schedule.DaySchedule](#) property), 144
[Comments](#) ([archetypal.template.schedule.WeekSchedule](#) property), 149
[Comments](#) ([archetypal.template.schedule.YearSchedule](#) property), 153
[Comments](#) ([archetypal.template.StructureInformation](#) property), 102
[Comments](#) ([archetypal.template.umi_base.UmiBase](#) property), 131
[Comments](#) ([archetypal.template.UmiSchedule](#) property), 98
[Comments](#) ([archetypal.template.VentilationSetting](#) property), 106
[Comments](#) ([archetypal.template.WindowConstruction](#) property), 115
[Comments](#) ([archetypal.template.WindowSetting](#) property), 119
[Comments](#) ([archetypal.template.ZoneConditioning](#) property), 65
[Comments](#) ([archetypal.template.ZoneConstructionSet](#) property), 128
[Comments](#) ([archetypal.template.ZoneDefinition](#) property), 124
[Comments](#) ([archetypal.template.ZoneLoad](#) property), 82
[Conditioning](#) ([archetypal.template.ZoneDefinition](#) property), 124
[Conductivity](#) ([archetypal.template.GasMaterial](#) property), 73
[Conductivity](#) ([archetypal.template.GlazingMaterial](#) property), 77
[Conductivity](#) ([archetypal.template.OpaqueMaterial](#) property), 93
[conductivity_at_temperature\(\)](#) ([archetypal.template.GasMaterial](#) method), 75
[config\(\)](#) (in module [archetypal.utils](#)), 193
[constant_schedule\(\)](#) ([archetypal.schedule.Schedule](#) class method), 183
[constant_schedule\(\)](#) ([archetypal.template.schedule.DaySchedule](#) class method), 145
[constant_schedule\(\)](#) ([archetypal.template.schedule.WeekSchedule](#) class method), 149
[constant_schedule\(\)](#) ([archetypal.template.schedule.YearSchedule](#) class method), 154
[constant_schedule\(\)](#) ([archetypal.template.UmiSchedule](#) class method), 97
[Construction](#) ([archetypal.template.WindowSetting](#) property), 120
[ConstructionBase](#) (class in [archetypal.template.constructions.base_construction](#)), 136
[Constructions](#) ([archetypal.template.ZoneDefinition](#) property), 124
[convert](#) ([archetypal.idfclass.IDF](#) property), 36
[CoolingCoeffOfPerf](#) ([archetypal.template.ZoneConditioning](#) property), 66
[CoolingFuelType](#) ([archetypal.template.ZoneConditioning](#) property), 66
[CoolingLimitType](#) ([archetypal.template.ZoneConditioning](#) property), 66
[CoolingSchedule](#) ([archetypal.template.ZoneConditioning](#) property), 66
[CoolingSetpoint](#) ([archetypal.template.ZoneConditioning](#) property), 65
[copy\(\)](#) ([archetypal.idfclass.IDF](#) method), 45
[copy\(\)](#) ([archetypal.zone_graph.ZoneGraph](#) method), 166
[copy_file\(\)](#) (in module [archetypal.utils](#)), 195
[copyidfobject\(\)](#) ([archetypal.idfclass.IDF](#) method), 39
[Core](#) ([archetypal.template.BuildingTemplate](#) property), 59
[core_graph](#) ([archetypal.zone_graph.ZoneGraph](#) property), 161
[Cost](#) ([archetypal.template.GasMaterial](#) property), 74
[Cost](#) ([archetypal.template.GlazingMaterial](#) property), 77
[Cost](#) ([archetypal.template.materials.material_base.MaterialBase](#) property), 133
[Cost](#) ([archetypal.template.OpaqueMaterial](#) property), 93
[Country](#) ([archetypal.template.BuildingTemplate](#) property), 60
[custom_processes](#) ([archetypal.idfclass.IDF](#) property), 35
[custom_profile\(\)](#) ([archetypal.idfclass.IDF](#) method), 48

D

[DataSource](#) ([archetypal.template.BuildingTemplate](#) property), 59
[DataSource](#) ([archetypal.template.constructions.base_construction.Construction](#) property), 136
[DataSource](#) ([archetypal.template.constructions.base_construction.LayeredConstruction](#) property), 138
[DataSource](#) ([archetypal.template.DomesticHotWaterSetting](#) property), 70
[DataSource](#) ([archetypal.template.GasMaterial](#) property), 74
[DataSource](#) ([archetypal.template.GlazingMaterial](#) property), 77

- DataSource (archetypal.template.materials.material_base property), 133
- DataSource (archetypal.template.OpaqueConstruction property), 89
- DataSource (archetypal.template.OpaqueMaterial property), 93
- DataSource (archetypal.template.schedule.DaySchedule property), 144
- DataSource (archetypal.template.schedule.WeekSchedule property), 149
- DataSource (archetypal.template.schedule.YearSchedule property), 153
- DataSource (archetypal.template.StructureInformation property), 102
- DataSource (archetypal.template.umi_base.UmiBase property), 131
- DataSource (archetypal.template.UmiSchedule property), 98
- DataSource (archetypal.template.VentilationSetting property), 106
- DataSource (archetypal.template.WindowConstruction property), 115
- DataSource (archetypal.template.WindowSetting property), 119
- DataSource (archetypal.template.ZoneConditioning property), 65
- DataSource (archetypal.template.ZoneConstructionSet property), 128
- DataSource (archetypal.template.ZoneDefinition property), 124
- DataSource (archetypal.template.ZoneLoad property), 82
- day_of_week_for_start_day (archetypal.idfclass.IDF property), 43
- DaylightMeshResolution (archetypal.template.ZoneDefinition property), 124
- DaylightWorkplaneHeight (archetypal.template.ZoneDefinition property), 125
- Days (archetypal.template.schedule.WeekSchedule property), 148
- DaySchedule (class in archetypal.template.schedule), 142
- DefaultWindowToWallRatio (archetypal.template.BuildingTemplate property), 59
- degree (archetypal.zone_graph.ZoneGraph property), 167
- Density (archetypal.template.GasMaterial property), 73
- Density (archetypal.template.GlazingMaterial property), 77
- Density (archetypal.template.OpaqueMaterial property), 93
- density_at_temperature() (archetypal.template.GasMaterial method), 74
- MessageID (archetypal.idfclass.IDF property), 36
- develop() (archetypal.template.schedule.DaySchedule method), 145
- develop() (archetypal.template.schedule.WeekSchedule method), 149
- develop() (archetypal.template.schedule.YearSchedule method), 154
- develop() (archetypal.template.UmiSchedule method), 98
- DimmingType (archetypal.template.ZoneLoad property), 82
- DirtFactor (archetypal.template.GlazingMaterial property), 77
- DisassemblyCarbon (archetypal.template.constructions.base_construction.ConstructionBase property), 136
- DisassemblyCarbon (archetypal.template.constructions.base_construction.LayeredConstruction property), 138
- DisassemblyCarbon (archetypal.template.OpaqueConstruction property), 89
- DisassemblyCarbon (archetypal.template.StructureInformation property), 102
- DisassemblyCarbon (archetypal.template.WindowConstruction property), 115
- DisassemblyEnergy (archetypal.template.constructions.base_construction.ConstructionBase property), 136
- DisassemblyEnergy (archetypal.template.constructions.base_construction.LayeredConstruction property), 138
- DisassemblyEnergy (archetypal.template.OpaqueConstruction property), 89
- DisassemblyEnergy (archetypal.template.StructureInformation property), 102
- DisassemblyEnergy (archetypal.template.WindowConstruction property), 115
- DomesticHotWater (archetypal.template.ZoneDefinition property), 124
- DomesticHotWaterSetting (class in archetypal.template), 69
- dominant_wall() (archetypal.template.OpaqueConstruction method), 88
- download_bld_window() (in module archetypal.dataportal), 189
- duplicate() (archetypal.template.BuildingTemplate method), 59

- duplicate() (archetypal.template.constructions.base_construction.ConstructionBase method), 136
- duplicate() (archetypal.template.constructions.base_construction.LayeredConstruction method), 138
- duplicate() (archetypal.template.DomesticHotWaterSetting method), 72
- duplicate() (archetypal.template.GasMaterial method), 73
- duplicate() (archetypal.template.GlazingMaterial method), 78
- duplicate() (archetypal.template.materials.material_base.MaterialBase method), 133
- duplicate() (archetypal.template.materials.material_layer.MaterialLayer method), 135
- duplicate() (archetypal.template.OpaqueConstruction method), 89
- duplicate() (archetypal.template.OpaqueMaterial method), 96
- duplicate() (archetypal.template.schedule.DaySchedule method), 143
- duplicate() (archetypal.template.schedule.WeekSchedule method), 148
- duplicate() (archetypal.template.schedule.YearSchedule method), 154
- duplicate() (archetypal.template.structure.MassRatio method), 141
- duplicate() (archetypal.template.StructureInformation method), 104
- duplicate() (archetypal.template.umi_base.UmiBase method), 131
- duplicate() (archetypal.template.UmiSchedule method), 98
- duplicate() (archetypal.template.VentilationSetting method), 108
- duplicate() (archetypal.template.WindowConstruction method), 113
- duplicate() (archetypal.template.WindowSetting method), 119
- duplicate() (archetypal.template.ZoneConditioning method), 69
- duplicate() (archetypal.template.ZoneConstructionSet method), 130
- duplicate() (archetypal.template.ZoneDefinition method), 124
- duplicate() (archetypal.template.ZoneLoad method), 82
- ## E
- EconomizerType (archetypal.template.ZoneConditioning property), 67
- edge_attr_dict_factory (archetypal.zone_graph.ZoneGraph attribute), 168
- edge_subgraph() (archetypal.zone_graph.ZoneGraph method), 168
- edges (archetypal.zone_graph.ZoneGraph property), 168
- EmbodiedCarbon (archetypal.template.GasMaterial property), 74
- EmbodiedCarbon (archetypal.template.GlazingMaterial property), 77
- EmbodiedCarbon (archetypal.template.materials.material_base.MaterialBase property), 133
- EmbodiedCarbon (archetypal.template.OpaqueMaterial property), 93
- EmbodiedEnergy (archetypal.template.GasMaterial property), 74
- EmbodiedEnergy (archetypal.template.GlazingMaterial property), 77
- EmbodiedEnergy (archetypal.template.materials.material_base.MaterialBase property), 134
- EmbodiedEnergy (archetypal.template.OpaqueMaterial property), 93
- energyplus_its (archetypal.idfclass.IDF property), 38
- epmacro (archetypal.idfclass.IDF property), 36
- epw (archetypal.idfclass.IDF property), 36
- EquipmentAvailabilitySchedule (archetypal.template.ZoneLoad property), 82
- EquipmentPowerDensity (archetypal.template.ZoneLoad property), 82
- equivalent_heat_capacity_per_unit_volume (archetypal.template.OpaqueConstruction property), 87
- expandobjects (archetypal.idfclass.IDF property), 36
- extend() (archetypal.template.BuildingTemplate method), 59
- extend() (archetypal.template.constructions.base_construction.ConstructionBase method), 137
- extend() (archetypal.template.constructions.base_construction.LayeredConstruction method), 138
- extend() (archetypal.template.DomesticHotWaterSetting method), 71
- extend() (archetypal.template.GasMaterial method), 74
- extend() (archetypal.template.GlazingMaterial method), 78
- extend() (archetypal.template.materials.material_base.MaterialBase method), 133
- extend() (archetypal.template.OpaqueConstruction method), 89
- extend() (archetypal.template.OpaqueMaterial method), 96

method), 94

`extend()` (archetypal.template.schedule.DaySchedule method), 145

`extend()` (archetypal.template.schedule.WeekSchedule method), 149

`extend()` (archetypal.template.schedule.YearSchedule method), 154

`extend()` (archetypal.template.StructureInformation method), 103

`extend()` (archetypal.template.umi_base.UmiBase method), 132

`extend()` (archetypal.template.UmiSchedule method), 99

`extend()` (archetypal.template.VentilationSetting method), 106

`extend()` (archetypal.template.WindowConstruction method), 115

`extend()` (archetypal.template.WindowSetting method), 119

`extend()` (archetypal.template.ZoneConditioning method), 66

`extend()` (archetypal.template.ZoneConstructionSet method), 128

`extend()` (archetypal.template.ZoneDefinition method), 124

`extend()` (archetypal.template.ZoneLoad method), 82

F

Facade (archetypal.template.ZoneConstructionSet property), 127

`file_version` (archetypal.idfclass.IDF property), 35

`filter_report_data()` (archetypal.reportdata.ReportData method), 191

`filter_tabular_data()` (archetypal.tabulardata.TabularData method), 192

`float_mean()` (archetypal.template.BuildingTemplate method), 59

`float_mean()` (archetypal.template.constructions.base_construction.ConstructionBase method), 137

`float_mean()` (archetypal.template.constructions.base_construction.LayeredConstruction method), 138

`float_mean()` (archetypal.template.DomesticHotWaterSetting method), 71

`float_mean()` (archetypal.template.GasMaterial method), 74

`float_mean()` (archetypal.template.GlazingMaterial method), 78

`float_mean()` (archetypal.template.materials.material_base.MaterialBase method), 133

`float_mean()` (archetypal.template.OpaqueConstruction method), 89

`float_mean()` (archetypal.template.OpaqueMaterial method), 94

`float_mean()` (archetypal.template.schedule.DaySchedule method), 145

`float_mean()` (archetypal.template.schedule.WeekSchedule method), 150

`float_mean()` (archetypal.template.schedule.YearSchedule method), 154

`float_mean()` (archetypal.template.StructureInformation method), 103

`float_mean()` (archetypal.template.umi_base.UmiBase method), 131

`float_mean()` (archetypal.template.UmiSchedule method), 99

`float_mean()` (archetypal.template.VentilationSetting method), 107

`float_mean()` (archetypal.template.WindowConstruction method), 115

`float_mean()` (archetypal.template.WindowSetting method), 120

`float_mean()` (archetypal.template.ZoneConditioning method), 66

`float_mean()` (archetypal.template.ZoneConstructionSet method), 128

`float_mean()` (archetypal.template.ZoneDefinition method), 124

`float_mean()` (archetypal.template.ZoneLoad method), 83

`float_round()` (in module archetypal.utils), 196

`FlowRatePerFloorArea` (archetypal.template.DomesticHotWaterSetting property), 70

`fresh_air_from_ideal_loads()` (archetypal.template.ZoneConditioning method), 68

`fresh_air_from_zone_sizes()` (archetypal.template.ZoneConditioning method), 68

`from_construction()` (archetypal.template.WindowSetting class method), 120

`from_dict()` (archetypal.template.BuildingTemplate class method), 60

`from_dict()` (archetypal.template.DomesticHotWaterSetting class method), 70

`method`), 70
`from_dict()` (`archetypal.template.GasMaterial` class method), 73
`from_dict()` (`archetypal.template.GlazingMaterial` class method), 80
`from_dict()` (`archetypal.template.OpaqueConstruction` class method), 88
`from_dict()` (`archetypal.template.OpaqueMaterial` class method), 95
`from_dict()` (`archetypal.template.schedule.DaySchedule` class method), 143
`from_dict()` (`archetypal.template.schedule.WeekSchedule` class method), 148
`from_dict()` (`archetypal.template.schedule.YearSchedule` class method), 157
`from_dict()` (`archetypal.template.schedule.YearSchedulePart` class method), 141
`from_dict()` (`archetypal.template.StructureInformation` class method), 103
`from_dict()` (`archetypal.template.VentilationSetting` class method), 107
`from_dict()` (`archetypal.template.WindowConstruction` class method), 112
`from_dict()` (`archetypal.template.WindowSetting` class method), 121
`from_dict()` (`archetypal.template.ZoneConditioning` class method), 67
`from_dict()` (`archetypal.template.ZoneConstructionSet` class method), 129
`from_dict()` (`archetypal.template.ZoneDefinition` class method), 125
`from_dict()` (`archetypal.template.ZoneLoad` class method), 83
`from_epbunch()` (`archetypal.schedule.Schedule` class method), 183
`from_epbunch()` (`archetypal.template.OpaqueConstruction` class method), 88
`from_epbunch()` (`archetypal.template.OpaqueMaterial` class method), 95
`from_epbunch()` (`archetypal.template.schedule.DaySchedule` class method), 142
`from_epbunch()` (`archetypal.template.schedule.WeekSchedule` class method), 148
`from_epbunch()` (`archetypal.template.schedule.YearSchedule` class method), 154
`from_epbunch()` (`archetypal.template.UmiSchedule` class method), 99
`from_epbunch()` (`archetypal.template.WindowConstruction` class method), 112
`from_epbunch()` (`archetypal.template.ZoneDefinition` class method), 126
`from_example_files()` (`archetypal.idfclass.IDF` class method), 34
`from_idf()` (`archetypal.template.BuildingTemplate` class method), 61
`from_idf()` (`archetypal.zone_graph.ZoneGraph` class method), 158
`from_idf_files()` (`archetypal.umi_template.UmiTemplateLibrary` class method), 55
`from_ref()` (`archetypal.template.WindowSetting` class method), 122
`from_shgc()` (`archetypal.template.WindowConstruction` class method), 112
`from_sql()` (`archetypal.tabulardata.TabularData` class method), 192
`from_sql_dict()` (`archetypal.reportdata.ReportData` class method), 190
`from_sqlite()` (`archetypal.reportdata.ReportData` class method), 190
`from_surface()` (`archetypal.template.WindowSetting` class method), 121
`from_values()` (`archetypal.schedule.Schedule` class method), 183
`from_values()` (`archetypal.template.schedule.DaySchedule` class method), 142
`from_values()` (`archetypal.template.schedule.WeekSchedule` class method), 150
`from_values()` (`archetypal.template.schedule.YearSchedule` class method), 154
`from_values()` (`archetypal.template.UmiSchedule` class method), 97
`from_zone()` (`archetypal.template.ZoneConditioning` class method), 68
`FromDay` (`archetypal.template.schedule.YearSchedulePart` property), 141
`FromMonth` (`archetypal.template.schedule.YearSchedulePart` property), 141

G

`gap_count` (`archetypal.template.WindowConstruction` property), 111
`GasMaterial` (class in `archetypal.template`), 72
`generic()` (`archetypal.template.OpaqueConstruction` class method), 89

`generic()` (*archetypal.template.OpaqueMaterial* class method), 94
`generic()` (*archetypal.template.structure.MassRatio* class method), 141
`generic()` (*archetypal.template.WindowSetting* class method), 120
`generic_internalmass()` (*archetypal.template.OpaqueConstruction* class method), 88
`get_days()` (*archetypal.template.schedule.WeekSchedule* class method), 148
`get_edge_data()` (*archetypal.zone_graph.ZoneGraph* method), 169
`get_equipment_list()` (*archetypal.template.ZoneConditioning* static method), 68
`get_ref()` (*archetypal.template.BuildingTemplate* method), 62
`get_ref()` (*archetypal.template.schedule.DaySchedule* method), 145
`get_ref()` (*archetypal.template.schedule.WeekSchedule* method), 150
`get_ref()` (*archetypal.template.schedule.YearSchedule* method), 154
`get_ref()` (*archetypal.template.UmiSchedule* method), 98
`get_schedule_epbunch()` (*archetypal.idfclass.IDF* method), 49
`get_schedule_type_limits_data_by_name()` (*archetypal.idfclass.IDF* method), 49
`get_schedule_type_limits_name()` (*archetypal.pal.schedule.Schedule* static method), 183
`get_schedule_type_limits_name()` (*archetypal.pal.template.schedule.DaySchedule* static method), 145
`get_schedule_type_limits_name()` (*archetypal.pal.template.schedule.WeekSchedule* static method), 150
`get_schedule_type_limits_name()` (*archetypal.pal.template.schedule.YearSchedule* static method), 155
`get_schedule_type_limits_name()` (*archetypal.pal.template.UmiSchedule* static method), 99
`get_unique()` (*archetypal.template.BuildingTemplate* method), 62
`get_unique()` (*archetypal.pal.template.constructions.base_construction.ConstructionBase* method), 137
`get_unique()` (*archetypal.pal.template.constructions.base_construction.LayerConstruction* method), 139
`get_unique()` (*archetypal.pal.template.DomesticHotWaterSetting* method), 71
`get_unique()` (*archetypal.pal.template.GasMaterial* method), 75
`get_unique()` (*archetypal.pal.template.GlazingMaterial* method), 78
`get_unique()` (*archetypal.pal.template.materials.material_base.MaterialBase* method), 133
`get_unique()` (*archetypal.pal.template.materials.material_layer.MaterialLayer* method), 135
`get_unique()` (*archetypal.pal.template.OpaqueConstruction* method), 90
`get_unique()` (*archetypal.pal.template.OpaqueMaterial* method), 94
`get_unique()` (*archetypal.pal.template.schedule.DaySchedule* method), 143
`get_unique()` (*archetypal.pal.template.schedule.WeekSchedule* method), 148
`get_unique()` (*archetypal.pal.template.schedule.YearSchedule* method), 157
`get_unique()` (*archetypal.pal.template.schedule.YearSchedulePart* method), 142
`get_unique()` (*archetypal.pal.template.structure.MassRatio* method), 141
`get_unique()` (*archetypal.pal.template.StructureInformation* method), 103
`get_unique()` (*archetypal.pal.template.umi_base.UmiBase* method), 132
`get_unique()` (*archetypal.pal.template.UmiSchedule* method), 98
`get_unique()` (*archetypal.pal.template.VentilationSetting* method), 107
`get_unique()` (*archetypal.pal.template.WindowConstruction* method), 115
`get_unique()` (*archetypal.pal.template.WindowSetting* method), 120
`get_unique()` (*archetypal.pal.template.ZoneConditioning* method), 66
`get_unique()` (*archetypal.pal.template.ZoneConstructionSet* method), 129
`get_unique()` (*archetypal.pal.template.ZoneDefinition* method), 124
`get_unique()` (*archetypal.pal.template.ZoneLoad* method), 83

[getextensibleindex\(\)](#) (*archetypal.idfclass.IDF method*), 39
[getiddgroupdict\(\)](#) (*archetypal.idfclass.IDF method*), 39
[getiddname\(\)](#) (*archetypal.idfclass.IDF method*), 35
[getobject\(\)](#) (*archetypal.idfclass.IDF method*), 40
[getshadingsurfaces\(\)](#) (*archetypal.idfclass.IDF method*), 40
[getsubsurfaces\(\)](#) (*archetypal.idfclass.IDF method*), 40
[getsurfaces\(\)](#) (*archetypal.idfclass.IDF method*), 40
[glazing_count](#) (*archetypal.template.WindowConstruction property*), 111
[GlazingMaterial](#) (*class in archetypal.template*), 76
[graph_attr_dict_factory](#) (*archetypal.zone_graph.ZoneGraph attribute*), 170
[Ground](#) (*archetypal.template.ZoneConstructionSet property*), 127

H

[has_edge\(\)](#) (*archetypal.zone_graph.ZoneGraph method*), 170
[has_node\(\)](#) (*archetypal.zone_graph.ZoneGraph method*), 171
[heat_balance\(\)](#) (*archetypal.template.WindowConstruction method*), 115
[heat_capacity](#) (*archetypal.template.materials.material_layer.MaterialLayer property*), 135
[heat_capacity_per_unit_wall_area](#) (*archetypal.template.OpaqueConstruction property*), 87
[HeatingCoeffOfPerf](#) (*archetypal.template.ZoneConditioning property*), 66
[HeatingFuelType](#) (*archetypal.template.ZoneConditioning property*), 65
[HeatingLimitType](#) (*archetypal.template.ZoneConditioning property*), 65
[HeatingSchedule](#) (*archetypal.template.ZoneConditioning property*), 65
[HeatingSetpoint](#) (*archetypal.template.ZoneConditioning property*), 65
[HeatRecoveryEfficiencyLatent](#) (*archetypal.template.ZoneConditioning property*), 67
[HeatRecoveryEfficiencySensible](#) (*archetypal.template.ZoneConditioning property*), 67
[HeatRecoveryType](#) (*archetypal.template.ZoneConditioning property*), 67
[HighLoadRatio](#) (*archetypal.template.structure.MassRatio property*), 141
[htm\(\)](#) (*archetypal.idfclass.IDF method*), 38

[id](#) (*archetypal.template.BuildingTemplate property*), 59
[id](#) (*archetypal.template.constructions.base_construction.ConstructionBase property*), 137
[id](#) (*archetypal.template.constructions.base_construction.LayeredConstruction property*), 139
[id](#) (*archetypal.template.DomesticHotWaterSetting property*), 71
[id](#) (*archetypal.template.GasMaterial property*), 75
[id](#) (*archetypal.template.GlazingMaterial property*), 78
[id](#) (*archetypal.template.materials.material_base.MaterialBase property*), 133
[id](#) (*archetypal.template.OpaqueConstruction property*), 90
[id](#) (*archetypal.template.OpaqueMaterial property*), 94
[id](#) (*archetypal.template.schedule.DaySchedule property*), 145
[id](#) (*archetypal.template.schedule.WeekSchedule property*), 150
[id](#) (*archetypal.template.schedule.YearSchedule property*), 155
[id](#) (*archetypal.template.StructureInformation property*), 103
[id](#) (*archetypal.template.umi_base.UmiBase property*), 131
[id](#) (*archetypal.template.UmiSchedule property*), 99
[id](#) (*archetypal.template.VentilationSetting property*), 107
[id](#) (*archetypal.template.WindowConstruction property*), 115
[id](#) (*archetypal.template.WindowSetting property*), 120
[id](#) (*archetypal.template.ZoneConditioning property*), 66
[id](#) (*archetypal.template.ZoneConstructionSet property*), 129
[id](#) (*archetypal.template.ZoneDefinition property*), 124
[id](#) (*archetypal.template.ZoneLoad property*), 83
[idd_index](#) (*archetypal.idfclass.IDF property*), 35
[idd_info](#) (*archetypal.idfclass.IDF property*), 35
[idd_version](#) (*archetypal.idfclass.IDF property*), 35
[iddname](#) (*archetypal.idfclass.IDF property*), 35
[IDF](#) (*class in archetypal.idfclass*), 33
[idfname](#) (*archetypal.idfclass.IDF property*), 36
[idfobjects](#) (*archetypal.idfclass.IDF property*), 35
[idfstr\(\)](#) (*archetypal.idfclass.IDF method*), 40
[idfversionupdater_dir](#) (*archetypal.idfclass.IDF property*), 37

IlluminanceTarget (archetypal.template.ZoneLoad property), 82
in_h() (archetypal.template.constructions.base_construction method), 139
in_h() (archetypal.template.OpaqueConstruction method), 90
in_h() (archetypal.template.WindowConstruction method), 116
in_h_c() (archetypal.template.constructions.base_construction.LayeredConstruction method), 140
in_h_c() (archetypal.template.OpaqueConstruction method), 90
in_h_c() (archetypal.template.WindowConstruction method), 116
in_h_simple() (archetypal.template.constructions.base_construction.LayeredConstruction method), 139
in_h_simple() (archetypal.template.OpaqueConstruction method), 90
in_h_simple() (archetypal.template.WindowConstruction method), 116
include (archetypal.idfclass.IDF property), 35
infer_insulation_layer() (archetypal.template.OpaqueConstruction method), 87
Infiltration (archetypal.template.VentilationSetting property), 106
info() (archetypal.zone_graph.ZoneGraph method), 161
initnew() (archetypal.idfclass.IDF method), 40
initread() (archetypal.idfclass.IDF method), 40
initreadtxt() (archetypal.idfclass.IDF method), 40
inside_emissivity (archetypal.template.constructions.base_construction.LayeredConstruction property), 140
inside_emissivity (archetypal.template.OpaqueConstruction property), 90
inside_emissivity (archetypal.template.WindowConstruction property), 111
InternalMassConstruction (archetypal.template.ZoneDefinition property), 125
InternalMassExposedPerFloorArea (archetypal.template.ZoneDefinition property), 125
intersect() (archetypal.idfclass.IDF method), 40
intersect_match() (archetypal.idfclass.IDF method), 40
IREmissivityBack (archetypal.template.GlazingMaterial property), 77
IREmissivityFront (archetypal.template.GlazingMaterial property), 77
IRTransmittance (archetypal.template.GlazingMaterial property), 78
is_core (archetypal.template.ZoneDefinition property), 125
is_directed() (archetypal.zone_graph.ZoneGraph method), 171
is_multigraph() (archetypal.zone_graph.ZoneGraph method), 171
is_part_of_conditioned_floor_area (archetypal.template.ZoneDefinition property), 125
is_part_of_total_floor_area (archetypal.template.ZoneDefinition property), 125
IsBoilerOn (archetypal.template.VentilationSetting property), 106
IsCoolingOn (archetypal.template.ZoneConditioning property), 66
IsEquipmentOn (archetypal.template.ZoneLoad property), 83
IsFacadeAdiabatic (archetypal.template.ZoneConstructionSet property), 128
IsGroundAdiabatic (archetypal.template.ZoneConstructionSet property), 128
IsHeatingOn (archetypal.template.ZoneConditioning property), 65
IsInfiltrationOn (archetypal.template.VentilationSetting property), 106
IsLightingOn (archetypal.template.ZoneLoad property), 83
IsMechVentOn (archetypal.template.ZoneConditioning property), 66
IsNatVentOn (archetypal.template.VentilationSetting property), 106
IsOn (archetypal.template.DomesticHotWaterSetting property), 70
IsPartitionAdiabatic (archetypal.template.ZoneConstructionSet property), 128
IsPeopleOn (archetypal.template.ZoneLoad property), 83
IsRoofAdiabatic (archetypal.template.ZoneConstructionSet property), 128
IsScheduledVentilationOn (archetypal.template.VentilationSetting property), 106
IsShadingSystemOn (archetypal.template.WindowSetting property), 120
IsSlabAdiabatic (archetypal.template.ZoneConstructionSet property), 128

- `pal.template.ZoneConstructionSet` (property), 128
- `IsVirtualPartition` (`archetypal.template.WindowSetting` property), 120
- `IsWindOn` (`archetypal.template.VentilationSetting` property), 106
- `IsZoneMixingOn` (`archetypal.template.WindowSetting` property), 120
- ## K
- `keep_data` (`archetypal.idfclass.IDF` property), 35
- `keep_data_err` (`archetypal.idfclass.IDF` property), 35
- ## L
- `LayeredConstruction` (class in `archetypal.template.constructions.base_construction`), 137
- `Layers` (`archetypal.template.constructions.base_construction.LayeredConstruction` attribute), 137
- `Layers` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 138
- `Layers` (`archetypal.template.OpaqueConstruction` property), 89
- `Layers` (`archetypal.template.WindowConstruction` property), 115
- `lcm()` (in module `archetypal.utils`), 197
- `length` (`archetypal.idfclass.IDF` property), 49
- `Lifespan` (`archetypal.template.BuildingTemplate` property), 59
- `LightingPowerDensity` (`archetypal.template.ZoneLoad` property), 82
- `LightsAvailabilitySchedule` (`archetypal.template.ZoneLoad` property), 82
- `load_umi_template()` (in module `archetypal.utils`), 195
- `Loads` (`archetypal.template.ZoneDefinition` property), 124
- `loads()` (`archetypal.umi_template.UmiTemplateLibrary` class method), 55
- `log()` (in module `archetypal.utils`), 194
- ## M
- `mapping()` (`archetypal.template.BuildingTemplate` method), 62
- `mapping()` (`archetypal.template.constructions.base_construction.LayeredConstruction` method), 137
- `mapping()` (`archetypal.template.constructions.base_construction.LayeredConstruction` method), 139
- `mapping()` (`archetypal.template.DomesticHotWaterSetting` method), 72
- `mapping()` (`archetypal.template.GasMaterial` method), 73
- `mapping()` (`archetypal.template.GlazingMaterial` method), 79
- `mapping()` (`archetypal.template.materials.material_base.MaterialBase` method), 134
- `mapping()` (`archetypal.template.materials.material_layer.MaterialLayer` method), 135
- `mapping()` (`archetypal.template.OpaqueConstruction` method), 89
- `mapping()` (`archetypal.template.OpaqueMaterial` method), 96
- `mapping()` (`archetypal.template.schedule.DaySchedule` method), 143
- `mapping()` (`archetypal.template.schedule.WeekSchedule` method), 148
- `mapping()` (`archetypal.template.schedule.YearSchedule` method), 157
- `mapping()` (`archetypal.template.schedule.YearSchedulePart` method), 142
- `mapping()` (`archetypal.template.structure.MassRatio` method), 141
- `mapping()` (`archetypal.template.StructureInformation` method), 103
- `mapping()` (`archetypal.template.umi_base.UmiBase` method), 132
- `mapping()` (`archetypal.template.UmiSchedule` method), 98
- `mapping()` (`archetypal.template.VentilationSetting` method), 108
- `mapping()` (`archetypal.template.WindowConstruction` method), 113
- `mapping()` (`archetypal.template.WindowSetting` method), 122
- `mapping()` (`archetypal.template.ZoneConditioning` method), 68
- `mapping()` (`archetypal.template.ZoneConstructionSet` method), 130
- `mapping()` (`archetypal.template.ZoneDefinition` method), 127
- `mapping()` (`archetypal.template.ZoneLoad` method), 84
- `mass_per_unit_area` (`archetypal.template.OpaqueConstruction` property), 87
- `MassRatio` (class in `archetypal.template.structure`), 140
- `MassRatios` (`archetypal.template.StructureInformation` property), 102
- `match()` (`archetypal.idfclass.IDF` method), 41
- `Material` (`archetypal.template.materials.material_layer.MaterialLayer` property), 134
- `Material` (`archetypal.template.structure.MassRatio` property), 141
- `MaterialBase` (class in `archetypal.template.materials.material_base`), 132
- `MaterialLayer` (class in `archetypal.template.materials.material_layer`), 134
- `max` (`archetypal.schedule.Schedule` property), 183
- `max` (`archetypal.template.schedule.DaySchedule` property), 143

erty), 145

max (archetypal.template.schedule.WeekSchedule property), 150

max (archetypal.template.schedule.YearSchedule property), 155

max (archetypal.template.UmiSchedule property), 99

MaxCoolFlow (archetypal.template.ZoneConditioning property), 65

MaxCoolingCapacity (archetypal.template.ZoneConditioning property), 65

MaxHeatFlow (archetypal.template.ZoneConditioning property), 65

MaxHeatingCapacity (archetypal.template.ZoneConditioning property), 65

mean (archetypal.schedule.Schedule property), 183

mean (archetypal.template.schedule.DaySchedule property), 145

mean (archetypal.template.schedule.WeekSchedule property), 150

mean (archetypal.template.schedule.YearSchedule property), 155

mean (archetypal.template.UmiSchedule property), 99

MechVentSchedule (archetypal.template.ZoneConditioning property), 67

meters (archetypal.idfclass.IDF property), 43

Meters (class in archetypal.idfclass), 53

min (archetypal.schedule.Schedule property), 183

min (archetypal.template.schedule.DaySchedule property), 145

min (archetypal.template.schedule.WeekSchedule property), 150

min (archetypal.template.schedule.YearSchedule property), 155

min (archetypal.template.UmiSchedule property), 99

MinFreshAirPerArea (archetypal.template.ZoneConditioning property), 65

MinFreshAirPerPerson (archetypal.template.ZoneConditioning property), 65

model (archetypal.idfclass.IDF property), 35

MoistureDiffusionResistance (archetypal.template.OpacityMaterial property), 94

molecular_weight (archetypal.template.GasMaterial property), 73

mtd_file (archetypal.idfclass.IDF property), 38

multiplier (archetypal.template.ZoneDefinition property), 125

N

name (archetypal.idfclass.IDF property), 37

Name (archetypal.schedule.Schedule property), 183

Name (archetypal.template.BuildingTemplate property), 59

Name (archetypal.template.constructions.base_construction.ConstructionBase property), 136

Name (archetypal.template.constructions.base_construction.LayeredConstruction property), 138

Name (archetypal.template.DomesticHotWaterSetting property), 70

Name (archetypal.template.GasMaterial property), 74

Name (archetypal.template.GlazingMaterial property), 77

Name (archetypal.template.materials.material_base.MaterialBase property), 133

Name (archetypal.template.OpaqueConstruction property), 89

Name (archetypal.template.OpaqueMaterial property), 94

Name (archetypal.template.schedule.DaySchedule property), 144

Name (archetypal.template.schedule.WeekSchedule property), 149

Name (archetypal.template.schedule.YearSchedule property), 153

Name (archetypal.template.StructureInformation property), 103

Name (archetypal.template.umi_base.UmiBase property), 131

Name (archetypal.template.UmiSchedule property), 98

Name (archetypal.template.VentilationSetting property), 106

Name (archetypal.template.WindowConstruction property), 115

Name (archetypal.template.WindowSetting property), 119

Name (archetypal.template.ZoneConditioning property), 66

Name (archetypal.template.ZoneConstructionSet property), 128

Name (archetypal.template.ZoneDefinition property), 124

Name (archetypal.template.ZoneLoad property), 82

name (archetypal.zone_graph.ZoneGraph property), 171

NatVentMaxOutdoorAirTemp (archetypal.template.VentilationSetting property), 106

NatVentMaxRelHumidity (archetypal.template.VentilationSetting property), 106

NatVentMinOutdoorAirTemp (archetypal.template.VentilationSetting property), 106

NatVentSchedule (archetypal.template.VentilationSetting property), 106

NatVentZoneTempSetpoint (archetypal-

pal.template.VentilationSetting (property), 107
nbunch_iter() (*archetypal.zone_graph.ZoneGraph* method), 171
neighbors() (*archetypal.zone_graph.ZoneGraph* method), 172
net_conditioned_building_area (*archetypal.idfclass.IDF* property), 38
new() (*archetypal.idfclass.IDF* method), 41
newidfobject() (*archetypal.idfclass.IDF* method), 48
node_attr_dict_factory (*archetypal.zone_graph.ZoneGraph* attribute), 172
node_dict_factory (*archetypal.zone_graph.ZoneGraph* attribute), 172
nodes (*archetypal.zone_graph.ZoneGraph* property), 172
NormalRatio (*archetypal.template.structure.MassRatio* property), 141
nrel_api_cbr_request() (in module *archetypal.dataportal*), 188
nrel_bcl_api_request() (in module *archetypal.dataportal*), 188
number_of_edges() (*archetypal.zone_graph.ZoneGraph* method), 174
number_of_nodes() (*archetypal.zone_graph.ZoneGraph* method), 174

O

OccupancySchedule (*archetypal.template.ZoneLoad* property), 82
occupants (*archetypal.template.ZoneDefinition* property), 125
OpaqueConstruction (class in *archetypal.template*), 86
OpaqueMaterial (class in *archetypal.template*), 92
open() (*archetypal.umi_template.UmiTemplateLibrary* class method), 55
open_err() (*archetypal.idfclass.IDF* method), 38
open_htm() (*archetypal.idfclass.IDF* method), 38
open_idf() (*archetypal.idfclass.IDF* method), 38
open_last_simulation() (*archetypal.idfclass.IDF* method), 38
open_mdd() (*archetypal.idfclass.IDF* method), 38
open_mtd() (*archetypal.idfclass.IDF* method), 38
openei_api_request() (in module *archetypal.dataportal*), 187
OperableArea (*archetypal.template.WindowSetting* property), 119
order() (*archetypal.zone_graph.ZoneGraph* method), 175
out_h() (*archetypal.template.constructions.base_construction.LayeredConstruction* method), 139
out_h() (*archetypal.template.OpaqueConstruction* method), 91
out_h() (*archetypal.template.WindowConstruction* method), 116

out_h_simple() (*archetypal.pal.template.constructions.base_construction.LayeredConstruction* method), 139
out_h_simple() (*archetypal.pal.template.OpaqueConstruction* method), 91
out_h_simple() (*archetypal.pal.template.WindowConstruction* method), 117
output_directory (*archetypal.idfclass.IDF* property), 36
output_prefix (*archetypal.idfclass.IDF* property), 37
output_suffix (*archetypal.idfclass.IDF* property), 35
OutputMeter (*archetypal.idfclass.Meters* attribute), 53
outputs (*archetypal.idfclass.IDF* property), 43
Outputs (class in *archetypal.idfclass*), 51
OutputVariable (*archetypal.idfclass.Variables* attribute), 53
outside_emissivity (*archetypal.pal.template.constructions.base_construction.LayeredConstruction* property), 140
outside_emissivity (*archetypal.pal.template.OpaqueConstruction* property), 91
outside_emissivity (*archetypal.pal.template.WindowConstruction* property), 111

P

parallel_process() (in module *archetypal.utils*), 197
Partition (*archetypal.template.ZoneConstructionSet* property), 128
partition_ratio (*archetypal.idfclass.IDF* property), 38
PartitionRatio (*archetypal.pal.template.BuildingTemplate* property), 60
PeopleDensity (*archetypal.template.ZoneLoad* property), 83
perim_graph (*archetypal.zone_graph.ZoneGraph* property), 161
Perimeter (*archetypal.template.BuildingTemplate* property), 59
plot() (*archetypal.schedule.Schedule* method), 183
plot() (*archetypal.template.schedule.DaySchedule* method), 145
plot() (*archetypal.template.schedule.WeekSchedule* method), 150
plot() (*archetypal.template.schedule.YearSchedule* method), 155
plot() (*archetypal.template.UmiSchedule* method), 99
plot2d() (*archetypal.schedule.Schedule* method), 184
plot2d() (*archetypal.template.schedule.DaySchedule* method), 146

- `plot2d()` (`archetypal.template.schedule.WeekSchedule` method), 151
`plot2d()` (`archetypal.template.schedule.YearSchedule` method), 155
`plot2d()` (`archetypal.template.UmiSchedule` method), 100
`plot_graph2d()` (`archetypal.zone_graph.ZoneGraph` method), 160
`plot_graph3d()` (`archetypal.zone_graph.ZoneGraph` method), 159
`popidfobject()` (`archetypal.idfclass.IDF` method), 41
`position` (`archetypal.idfclass.IDF` property), 37
`predecessors` (`archetypal.template.BuildingTemplate` property), 59
`predecessors` (`archetypal.template.constructions.base_construction.ComplexBase` property), 137
`predecessors` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 139
`predecessors` (`archetypal.template.DomesticHotWaterSetting` property), 71
`predecessors` (`archetypal.template.GasMaterial` property), 75
`predecessors` (`archetypal.template.GlazingMaterial` property), 78
`predecessors` (`archetypal.template.materials.material_base.MaterialBase` property), 134
`predecessors` (`archetypal.template.OpaqueConstruction` property), 91
`predecessors` (`archetypal.template.OpaqueMaterial` property), 94
`predecessors` (`archetypal.template.schedule.DaySchedule` property), 147
`predecessors` (`archetypal.template.schedule.WeekSchedule` property), 152
`predecessors` (`archetypal.template.schedule.YearSchedule` property), 156
`predecessors` (`archetypal.template.StructureInformation` property), 103
`predecessors` (`archetypal.template.umi_base.UmiBase` property), 131
`predecessors` (`archetypal.template.UmiSchedule` property), 101
`predecessors` (`archetypal.template.VentilationSetting` property), 107
`predecessors` (`archetypal.template.WindowConstruction` property), 117
`predecessors` (`archetypal.template.WindowSetting` property), 120
`predecessors` (`archetypal.template.ZoneConditioning` property), 66
`predecessors` (`archetypal.template.ZoneConstructionSet` property), 129
`predecessors` (`archetypal.template.ZoneDefinition` property), 124
`predecessors` (`archetypal.template.ZoneLoad` property), 83
`prep_outputs` (`archetypal.idfclass.IDF` property), 36
`printidf()` (`archetypal.idfclass.IDF` method), 41
`process_results()` (`archetypal.idfclass.IDF` method), 46
- ## Q
- `quantity` (`archetypal.template.schedule.DaySchedule` property), 147
`quantity` (`archetypal.template.schedule.WeekSchedule` property), 152
`quantity` (`archetypal.template.schedule.YearSchedule` property), 156
`quantity` (`archetypal.template.UmiSchedule` property), 97
- ## R
- `r_factor` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 139
`r_factor` (`archetypal.template.OpaqueConstruction` property), 91
`r_factor` (`archetypal.template.WindowConstruction` property), 111
`r_value` (`archetypal.template.constructions.base_construction.LayeredConstruction` property), 138
`r_value` (`archetypal.template.materials.material_layer.MaterialLayer` property), 135
`r_value` (`archetypal.template.OpaqueConstruction` property), 87
`r_value` (`archetypal.template.WindowConstruction` property), 111
`random()` (`archetypal.template.schedule.DaySchedule` class method), 147
`random()` (`archetypal.template.schedule.WeekSchedule` class method), 152
`random()` (`archetypal.template.schedule.YearSchedule` class method), 156
`random()` (`archetypal.template.UmiSchedule` class method), 97
`read()` (`archetypal.idfclass.IDF` method), 34
`readvars` (`archetypal.idfclass.IDF` property), 36
`recursive_len()` (in module `archetypal.utils`), 197

`reduced_model()` (archetypal.template.BuildingTemplate class method), 61
`remove_edge()` (archetypal.zone_graph.ZoneGraph method), 175
`remove_edges_from()` (archetypal.zone_graph.ZoneGraph method), 176
`remove_node()` (archetypal.zone_graph.ZoneGraph method), 176
`remove_nodes_from()` (archetypal.zone_graph.ZoneGraph method), 177
`removeextensibles()` (archetypal.idfclass.IDF method), 41
`removeidfobject()` (archetypal.idfclass.IDF method), 48
`removeidfobjects()` (archetypal.idfclass.IDF method), 49
`rename()` (archetypal.idfclass.IDF method), 50
`rename()` (archetypal.template.BuildingTemplate method), 60
`rename()` (archetypal.template.constructions.base_construction.ConstructionBase method), 137
`rename()` (archetypal.template.constructions.base_construction.RoofLayerConstruction method), 139
`rename()` (archetypal.template.DomesticHotWaterSetting method), 71
`rename()` (archetypal.template.GasMaterial method), 75
`rename()` (archetypal.template.GlazingMaterial method), 78
`rename()` (archetypal.template.materials.material_base.MaterialBase method), 134
`rename()` (archetypal.template.OpaqueConstruction method), 91
`rename()` (archetypal.template.OpaqueMaterial method), 94
`rename()` (archetypal.template.schedule.DaySchedule method), 147
`rename()` (archetypal.template.schedule.WeekSchedule method), 152
`rename()` (archetypal.template.schedule.YearSchedule method), 156
`rename()` (archetypal.template.StructureInformation method), 103
`rename()` (archetypal.template.umi_base.UmiBase method), 131
`rename()` (archetypal.template.UmiSchedule method), 101
`rename()` (archetypal.template.VentilationSetting method), 107
`rename()` (archetypal.template.WindowConstruction method), 117
`rename()` (archetypal.template.WindowSetting method), 120
`rename()` (archetypal.template.ZoneConditioning method), 66
`rename()` (archetypal.template.ZoneConstructionSet method), 129
`rename()` (archetypal.template.ZoneDefinition method), 125
`rename()` (archetypal.template.ZoneLoad method), 83
`replace()` (archetypal.schedule.Schedule method), 183
`replace()` (archetypal.template.schedule.DaySchedule method), 147
`replace()` (archetypal.template.schedule.WeekSchedule method), 152
`replace()` (archetypal.template.schedule.YearSchedule method), 156
`replace()` (archetypal.template.UmiSchedule method), 101
`replace_component()` (archetypal.umi_template.UmiTemplateLibrary method), 56
`resistivity` (archetypal.template.materials.material_layer.MaterialLayer property), 85
`resize()` (archetypal.idfclass.IDF method), 49
`RootLayerConstruction` (archetypal.template.ZoneConstructionSet property), 128
`rotate()` (archetypal.idfclass.IDF method), 41
`rotate()` (in module archetypal.utils), 197
`Roughness` (archetypal.template.OpaqueMaterial property), 93
`run()` (archetypal.idfclass.IDF method), 41

S

`save()` (archetypal.idfclass.IDF method), 45
`save()` (archetypal.umi_template.UmiTemplateLibrary method), 55
`saveas()` (archetypal.idfclass.IDF method), 45
`savecopy()` (archetypal.idfclass.IDF method), 45
`scale()` (archetypal.idfclass.IDF method), 42
`scale()` (archetypal.schedule.Schedule method), 183
`scale()` (archetypal.template.schedule.DaySchedule method), 147
`scale()` (archetypal.template.schedule.WeekSchedule method), 152
`scale()` (archetypal.template.schedule.YearSchedule method), 156
`scale()` (archetypal.template.UmiSchedule method), 101
`Schedule` (archetypal.template.schedule.YearSchedulePart property), 141
`Schedule` (class in archetypal.schedule), 182
`ScheduledVentilationAch` (archetypal.template.VentilationSetting property), 107
`ScheduledVentilationSchedule` (archetypal.template.VentilationSetting property), 106

ScheduledVentilationSetpoint	(archetypal.template.VentilationSetting property), 107	pal.template.GlazingMaterial	property), 79
schedules_dict	(archetypal.idfclass.IDF property), 43	SolarTransmittance	(archetypal.template.GlazingMaterial property), 79
series	(archetypal.schedule.Schedule property), 183	space_cooling_profile()	(archetypal.idfclass.IDF method), 47
series	(archetypal.template.schedule.DaySchedule property), 147	space_heating_profile()	(archetypal.idfclass.IDF method), 47
series	(archetypal.template.schedule.WeekSchedule property), 152	specific_heat	(archetypal.template.GasMaterial property), 73
series	(archetypal.template.schedule.YearSchedule property), 156	specific_heat	(archetypal.template.materials.material_layer.MaterialLayer property), 135
series	(archetypal.template.UmiSchedule property), 101	specific_heat	(archetypal.template.OpaqueConstruction property), 87
service_water_heating_profile()	(archetypal.idfclass.IDF method), 47	specific_heat_at_temperature()	(archetypal.template.GasMaterial method), 75
set_wwr()	(archetypal.idfclass.IDF method), 50	SpecificHeat	(archetypal.template.OpaqueMaterial property), 93
setidd()	(archetypal.idfclass.IDF method), 35	sql()	(archetypal.idfclass.IDF method), 37
setiddname()	(archetypal.idfclass.IDF method), 34	sql_file	(archetypal.idfclass.IDF property), 38
ShadingSystemAvailabilitySchedule	(archetypal.template.WindowSetting property), 120	startDate	(archetypal.schedule.Schedule property), 183
ShadingSystemSetpoint	(archetypal.template.WindowSetting property), 119	startDate	(archetypal.template.schedule.DaySchedule property), 147
ShadingSystemTransmittance	(archetypal.template.WindowSetting property), 119	startDate	(archetypal.template.schedule.WeekSchedule property), 152
ShadingSystemType	(archetypal.template.WindowSetting property), 119	startDate	(archetypal.template.schedule.YearSchedule property), 156
ShadingType	(class in archetypal.template.constructions.window_construction), 158	startDate	(archetypal.template.UmiSchedule property), 101
shgc()	(archetypal.template.WindowConstruction method), 114	stat_can_geo_request()	(in module archetypal.dataportal), 189
sim_id	(archetypal.idfclass.IDF property), 37	stat_can_request()	(in module archetypal.dataportal), 188
sim_info	(archetypal.idfclass.IDF property), 37	Structure	(archetypal.template.BuildingTemplate property), 59
sim_timestamp	(archetypal.idfclass.IDF property), 37	StructureInformation	(class in archetypal.template), 101
simulate()	(archetypal.idfclass.IDF method), 44	subgraph()	(archetypal.zone_graph.ZoneGraph method), 178
simulation_dir	(archetypal.idfclass.IDF property), 43	SubstitutionRatePattern	(archetypal.template.GasMaterial property), 74
simulation_files	(archetypal.idfclass.IDF property), 42	SubstitutionRatePattern	(archetypal.template.GlazingMaterial property), 77
size()	(archetypal.zone_graph.ZoneGraph method), 177	SubstitutionRatePattern	(archetypal.template.materials.material_base.MaterialBase property), 134
Slab	(archetypal.template.ZoneConstructionSet property), 128	SubstitutionRatePattern	(archetypal.template.OpaqueMaterial property), 94
solar_reflectance_index	(archetypal.template.OpaqueConstruction property), 87	SubstitutionTimestep	(archetypal.template.OpaqueMaterial property), 94
solar_transmittance	(archetypal.template.WindowConstruction property), 112		
SolarAbsorptance	(archetypal.template.OpaqueMaterial property), 93		
SolarReflectanceBack	(archetypal.template.GlazingMaterial property), 79		
SolarReflectanceFront	(archetypal.template.GlazingMaterial property), 79		

pal.template.GasMaterial property), 74
 SubstitutionTimestep (archetypal.template.GlazingMaterial property), 77
 SubstitutionTimestep (archetypal.template.materials.material_base.MaterialBase property), 134
 SubstitutionTimestep (archetypal.template.OpaqueMaterial property), 94
T
 tabula_available_buildings() (in module archetypal.dataportal), 186
 tabula_building_details_sheet() (in module archetypal.dataportal), 186
 tabula_system() (in module archetypal.dataportal), 187
 tabula_system_request() (in module archetypal.dataportal), 187
 temperature_profile() (archetypal.template.WindowConstruction method), 114
 template_complexity_reduction() (archetypal.umi_template.UmiTemplateLibrary static method), 55
 ThermalEmittance (archetypal.template.OpaqueMaterial property), 93
 Thickness (archetypal.template.materials.material_layer.MaterialLayer property), 135
 thickness (archetypal.template.WindowConstruction property), 112
 time_constant_per_unit_area (archetypal.template.OpaqueConstruction property), 87
 timeit() (in module archetypal.utils), 196
 to_dict() (archetypal.template.BuildingTemplate method), 61
 to_dict() (archetypal.template.constructions.base_construction.ConstructionBase method), 137
 to_dict() (archetypal.template.constructions.base_construction.ConstructionBase method), 139
 to_dict() (archetypal.template.DomesticHotWaterSetting method), 70
 to_dict() (archetypal.template.GasMaterial method), 73
 to_dict() (archetypal.template.GlazingMaterial method), 79
 to_dict() (archetypal.template.materials.material_base.MaterialBase method), 134
 to_dict() (archetypal.template.materials.material_layer.MaterialLayer method), 135
 to_dict() (archetypal.template.OpaqueConstruction method), 89
 to_dict() (archetypal.template.OpaqueMaterial method), 95
 to_dict() (archetypal.template.schedule.DaySchedule method), 143
 to_dict() (archetypal.template.schedule.WeekSchedule method), 148
 to_dict() (archetypal.template.schedule.YearSchedule method), 157
 to_dict() (archetypal.template.schedule.YearSchedulePart method), 142
 to_dict() (archetypal.template.structure.MassRatio method), 141
 to_dict() (archetypal.template.StructureInformation method), 103
 to_dict() (archetypal.template.umi_base.UmiBase method), 131
 to_dict() (archetypal.template.UmiSchedule method), 98
 to_dict() (archetypal.template.VentilationSetting method), 108
 to_dict() (archetypal.template.WindowConstruction method), 113
 to_dict() (archetypal.template.WindowSetting method), 121
 to_dict() (archetypal.template.ZoneConditioning method), 68
 to_dict() (archetypal.template.ZoneConstructionSet method), 130
 to_dict() (archetypal.template.ZoneDefinition method), 125
 to_dict() (archetypal.template.ZoneLoad method), 84
 to_dict() (archetypal.umi_template.UmiTemplateLibrary method), 56
 to_directed() (archetypal.zone_graph.ZoneGraph method), 179
 to_directed_class() (archetypal.zone_graph.ZoneGraph method), 179
 to_epbunch() (archetypal.template.GasMaterial method), 73
 to_epbunch() (archetypal.template.GlazingMaterial method), 79
 to_epbunch() (archetypal.template.materials.material_layer.MaterialLayer method), 135
 to_epbunch() (archetypal.template.OpaqueConstruction method), 91
 to_epbunch() (archetypal.template.OpaqueMaterial method), 96
 to_epbunch() (archetypal.template.schedule.DaySchedule method), 143

- `to_epbunch()` (archetypal.template.schedule.WeekSchedule method), 152
- `to_epbunch()` (archetypal.template.schedule.YearSchedule method), 157
- `to_epbunch()` (archetypal.template.VentilationSetting method), 108
- `to_epbunch()` (archetypal.template.WindowConstruction method), 113
- `to_epbunch()` (archetypal.template.ZoneConditioning method), 68
- `to_epbunch()` (archetypal.template.ZoneLoad method), 84
- `to_json()` (archetypal.umi_template.UmiTemplateLibrary method), 56
- `to_obj()` (archetypal.idfclass.IDF method), 42
- `to_ref()` (archetypal.template.BuildingTemplate method), 60
- `to_ref()` (archetypal.template.constructions.base_construction.ConstructionBase method), 137
- `to_ref()` (archetypal.template.constructions.base_construction.LayeredConstruction method), 139
- `to_ref()` (archetypal.template.DomesticHotWaterSetting method), 71
- `to_ref()` (archetypal.template.GasMaterial method), 75
- `to_ref()` (archetypal.template.GlazingMaterial method), 78
- `to_ref()` (archetypal.template.materials.material_base.MaterialBase method), 134
- `to_ref()` (archetypal.template.OpaqueConstruction method), 91
- `to_ref()` (archetypal.template.OpaqueMaterial method), 95
- `to_ref()` (archetypal.template.schedule.DaySchedule method), 143
- `to_ref()` (archetypal.template.schedule.WeekSchedule method), 148
- `to_ref()` (archetypal.template.schedule.YearSchedule method), 157
- `to_ref()` (archetypal.template.StructureInformation method), 103
- `to_ref()` (archetypal.template.umi_base.UmiBase method), 131
- `to_ref()` (archetypal.template.UmiSchedule method), 98
- `to_ref()` (archetypal.template.VentilationSetting method), 107
- `to_ref()` (archetypal.template.WindowConstruction method), 117
- `to_ref()` (archetypal.template.WindowSetting method), 120
- `to_ref()` (archetypal.template.ZoneConditioning method), 66
- `to_ref()` (archetypal.template.ZoneConstructionSet method), 129
- `to_ref()` (archetypal.template.ZoneDefinition method), 125
- `to_ref()` (archetypal.template.ZoneLoad method), 83
- `to_undirected()` (archetypal.zone_graph.ZoneGraph method), 179
- `to_undirected_class()` (archetypal.zone_graph.ZoneGraph method), 180
- `to_year_week_day()` (archetypal.schedule.Schedule method), 185
- `to_year_week_day()` (archetypal.template.schedule.DaySchedule method), 147
- `to_year_week_day()` (archetypal.template.schedule.WeekSchedule method), 152
- `to_year_week_day()` (archetypal.template.schedule.YearSchedule method), 157
- `to_year_week_day()` (archetypal.template.UmiSchedule method), 101
- `ToDay` (archetypal.template.schedule.YearSchedulePart property), 141
- `ToMonth` (archetypal.template.schedule.YearSchedulePart property), 141
- `top()` (in module archetypal.utils), 195
- `total_building_area` (archetypal.idfclass.IDF property), 38
- `total_thickness` (archetypal.template.OpaqueConstruction property), 87
- `translate()` (archetypal.idfclass.IDF method), 42
- `translate_to_origin()` (archetypal.idfclass.IDF method), 42
- `TransportCarbon` (archetypal.template.GasMaterial property), 74
- `TransportCarbon` (archetypal.template.GlazingMaterial property), 77
- `TransportCarbon` (archetypal.template.materials.material_base.MaterialBase property), 134
- `TransportCarbon` (archetypal.template.OpaqueMaterial property), 94
- `TransportDistance` (archetypal.template.GasMaterial property), 74
- `TransportDistance` (archetypal.template.GlazingMaterial property), 78
- `TransportDistance` (archetypal.template.materials.material_base.MaterialBase property), 134

[property](#)), 134
 TransportDistance (archetypal.template.OpaqueMaterial property), 94
 TransportEnergy (archetypal.template.GasMaterial property), 74
 TransportEnergy (archetypal.template.GlazingMaterial property), 78
 TransportEnergy (archetypal.template.materials.material_base.MaterialBase property), 134
 TransportEnergy (archetypal.template.OpaqueMaterial property), 94
 Type (archetypal.schedule.Schedule property), 182
 Type (archetypal.template.GasMaterial property), 73
 Type (archetypal.template.schedule.DaySchedule property), 144
 Type (archetypal.template.schedule.WeekSchedule property), 149
 Type (archetypal.template.schedule.YearSchedule property), 153
 Type (archetypal.template.UmiSchedule property), 98
 Type (archetypal.template.WindowSetting property), 120

U

u_factor (archetypal.template.constructions.base_construction.LayeredConstruction property), 140
 u_factor (archetypal.template.OpaqueConstruction property), 91
 u_factor (archetypal.template.WindowConstruction property), 117
 u_value (archetypal.template.constructions.base_construction.LayeredConstruction property), 139
 u_value (archetypal.template.materials.material_layer.MaterialLayer property), 135
 u_value (archetypal.template.OpaqueConstruction property), 91
 u_value (archetypal.template.WindowConstruction property), 117
 UmiBase (class in archetypal.template.umi_base), 131
 UmiSchedule (class in archetypal.template), 97
 UmiTemplateLibrary (class in archetypal.umi_template), 54
 unconditioned_building_area (archetypal.idfclass.IDF property), 38
 unique_components() (archetypal.umi_template.UmiTemplateLibrary method), 56
 update() (archetypal.zone_graph.ZoneGraph method), 180
 update_components_list() (archetypal.umi_template.UmiTemplateLibrary method), 57
 upgrade() (archetypal.idfclass.IDF method), 46

V

validate() (archetypal.template.BuildingTemplate method), 62
 validate() (archetypal.template.constructions.base_construction.Construction method), 136
 validate() (archetypal.template.constructions.base_construction.LayeredConstruction method), 139
 validate() (archetypal.template.DomesticHotWaterSetting method), 72
 validate() (archetypal.template.GasMaterial method), 75
 validate() (archetypal.template.GlazingMaterial method), 78
 validate() (archetypal.template.materials.material_base.MaterialBase method), 134
 validate() (archetypal.template.OpaqueConstruction method), 91
 validate() (archetypal.template.OpaqueMaterial method), 96
 validate() (archetypal.template.schedule.DaySchedule method), 147
 validate() (archetypal.template.schedule.WeekSchedule method), 152
 validate() (archetypal.template.schedule.YearSchedule method), 157
 validate() (archetypal.template.StructureInformation method), 103
 validate() (archetypal.template.umi_base.UmiBase method), 132
 validate() (archetypal.template.UmiSchedule method), 98
 validate() (archetypal.template.VentilationSetting method), 108
 validate() (archetypal.template.WindowConstruction method), 113
 validate() (archetypal.template.WindowSetting method), 122
 validate() (archetypal.template.ZoneConditioning method), 68
 validate() (archetypal.template.ZoneConstructionSet method), 130
 validate() (archetypal.template.ZoneDefinition method), 127
 validate() (archetypal.template.ZoneLoad method), 84
 validate() (archetypal.umi_template.UmiTemplateLibrary method), 55
 Values (archetypal.schedule.Schedule property), 182
 Values (archetypal.template.schedule.DaySchedule property), 144
 Values (archetypal.template.schedule.WeekSchedule property), 149

- Values** (*archetypal.template.schedule.YearSchedule property*), 153
Values (*archetypal.template.UmiSchedule property*), 98
variables (*archetypal.idfclass.IDF property*), 43
Variables (*class in archetypal.idfclass*), 53
Ventilation (*archetypal.template.ZoneDefinition property*), 124
VentilationSetting (*class in archetypal.template*), 104
VentilationType (*archetypal.template.VentilationSetting property*), 107
verbose (*archetypal.idfclass.IDF property*), 36
Version (*archetypal.template.BuildingTemplate property*), 60
view_model() (*archetypal.idfclass.IDF method*), 42
viscosity_at_temperature() (*archetypal.template.GasMaterial method*), 75
visible_transmittance (*archetypal.template.WindowConstruction property*), 112
VisibleAbsorptance (*archetypal.template.OpaqueMaterial property*), 93
VisibleReflectanceBack (*archetypal.template.GlazingMaterial property*), 78
VisibleReflectanceFront (*archetypal.template.GlazingMaterial property*), 78
VisibleTransmittance (*archetypal.template.GlazingMaterial property*), 78
volume (*archetypal.template.VentilationSetting property*), 107
volume (*archetypal.template.ZoneConstructionSet property*), 128
volume (*archetypal.template.ZoneDefinition property*), 125
volume (*archetypal.template.ZoneLoad property*), 83
- W**
WaterSchedule (*archetypal.template.DomesticHotWaterSetting property*), 70
WaterSupplyTemperature (*archetypal.template.DomesticHotWaterSetting property*), 70
WaterTemperatureInlet (*archetypal.template.DomesticHotWaterSetting property*), 70
WeekSchedule (*class in archetypal.template.schedule*), 148
weighted_mean() (*in module archetypal.utils*), 194
- whole_building()** (*archetypal.template.DomesticHotWaterSetting class method*), 72
width (*archetypal.idfclass.IDF property*), 49
WindowConstruction (*class in archetypal.template*), 110
Windows (*archetypal.template.BuildingTemplate property*), 59
Windows (*archetypal.template.ZoneDefinition property*), 125
WindowSetting (*class in archetypal.template*), 117
WindowType (*class in archetypal.template.constructions.window_construction*), 158
wwr() (*archetypal.idfclass.IDF method*), 46
- Y**
YearFrom (*archetypal.template.BuildingTemplate property*), 60
YearSchedule (*class in archetypal.template.schedule*), 153
YearSchedulePart (*class in archetypal.template.schedule*), 141
YearTo (*archetypal.template.BuildingTemplate property*), 60
- Z**
zone_surfaces (*archetypal.template.ZoneDefinition property*), 125
ZoneConditioning (*class in archetypal.template*), 62
ZoneConstructionSet (*class in archetypal.template*), 127
ZoneDefinition (*class in archetypal.template*), 122
ZoneGraph (*class in archetypal.zone_graph*), 158
ZoneLoad (*class in archetypal.template*), 80
ZoneMixingAvailabilitySchedule (*archetypal.template.WindowSetting property*), 120
ZoneMixingDeltaTemperature (*archetypal.template.WindowSetting property*), 120
ZoneMixingFlowRate (*archetypal.template.WindowSetting property*), 120