# archetypal Documentation

***Release 2.17***

**Samuel Letellier-Duchesne**

**Sep 15, 2023**

# GETTING STARTED

1 Description 3

2 Indices and tables 37

*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.

CHAPTER

ONE

# 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 foreword. Therefore, all the modules are discoverable and can be imported independently.

## 1.1 Installation

### 1.1.1 Requirements

> **Warning:** Archetypal is most compatible with energyplus v9.2.0 (download here at the bottom of the page). although it will parse, convert and simulate other versions (older or newer).

EnergyPlus should be installed in it's default location. On Windows that would be in *C:\EnergyPlusV9-2-0* and on MacOS that would be in */Applications/EnergyPlus-9-2-0*. For custom install locations, see *Creating an Umi template*

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 (and above). 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 installEnergyPlus 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 pacakge 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.

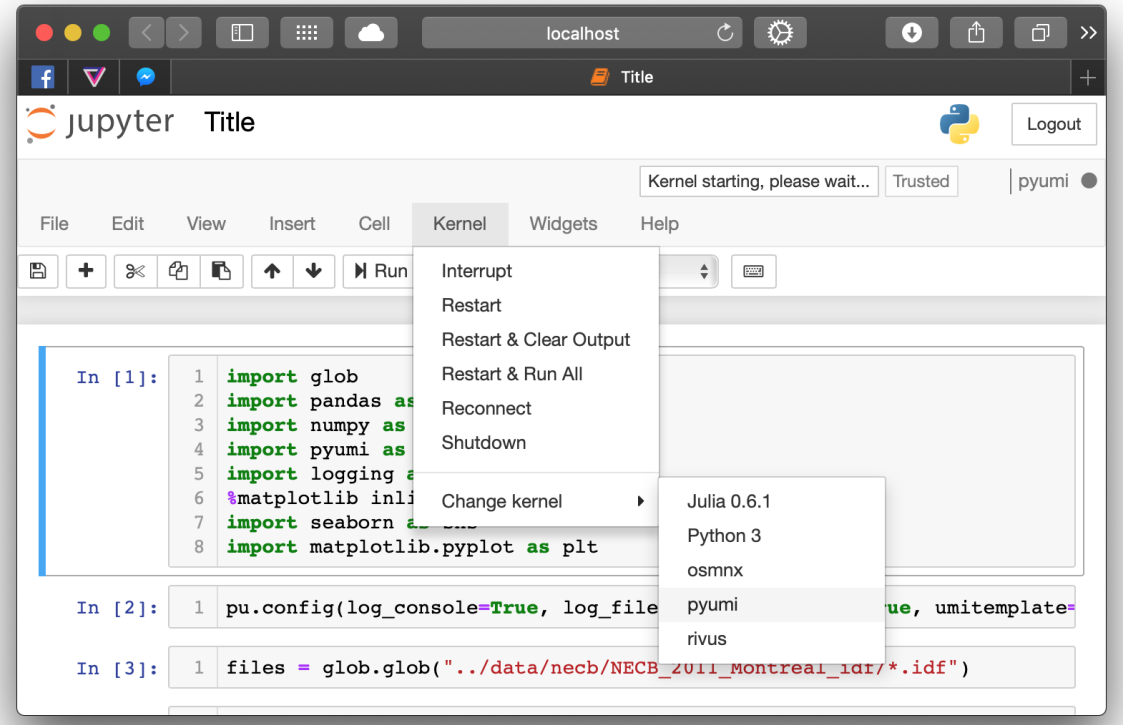



Fig. 1.1: choosing the correct kernel in a jupyter notebook. In the *kernel* menu, select *Change Kernel* and select the appropriate virtual env created earlier (*archetypal* in this case).

### 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.

### 1.3.1 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,
    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'),
```

(continues on next page)

(continued from previous page)

```
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'),
```

(continues on next page)

(continued from previous page)

```
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.

### 1.3.2 Clearing the cache

To clear the cache, invoke *clear_cache*:

## 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 be 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/BasicsFiles/
↪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 IDF
>>> idf = IDF.from_example_files("AdultEducationCenter.idf", epw="USA_IL_Chicago-OHare.
↪Intl.AP.725300_TMY3.epw")  # IDF load
```

2. Simulate the file

The model must be simulated because the BuildingTemplate.from_idf method uses the sqlite database generated by EnergyPlus.

```
>>> idf.simulate()
<IDF object AdultEducationCenter.idf
at /Applications/EnergyPlus-9-2-0/ExampleFiles/BasicsFiles/AdultEducationCenter.idf
    Version 9.2.0
Simulation Info:
| SimulationIndex       | 1                                                        |
| EnergyPlusVersion     | EnergyPlus, Version 9.2.0-921312fa1d, YMD=2023.01.28 18:44 |
| TimeStamp             | YMD=2023.01.28 18:44                                     |
| NumTimestepsPerHour   | 4                                                        |
| Completed             | 1                                                        |
| CompletedSuccessfully | 1                                                        |
    Files at 'cache/941af560028252d7311d572b9c84cee6/f79de785e8989c884dca20f1dca08c1f'>
```

3. Create a BuildingTemplate Object

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

4. 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 IDF
>>> idf = IDF("in.idf)  # in.idf must in the current directory.
```

You can also load on of the example files by name.

```
>>> from archetypal import IDF
>>> idf = IDF.from_example_files("AdultEducationCenter.idf")
```

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.outputs.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(epw=weather)
```

**Hint:** Caching system.

When running EnergyPlus simulations, a caching system is 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.

## 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(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("tests/input_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,
>>>         epw=str(epw),
>>>         expandobjects=True,
>>>         verbose=True,
>>>         design_day=True,
>>>         simulate=True,
>>>     )
>>>     for k, file in idfs.file.to_dict().items()
>>> }
```

We also define a generic function that takes the keyword arguments defined previously and loads, simulates and returns the SQL file path of the model.

```
def load_and_simulate(**kwargs):
    """Load IDF model, simulate, and return sql_file path."""
    return IDF(**kwargs).simulate().sql_file
```

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

```
>>> sql_files =  parallel_process(rundict, load_and_simulate, use_kwargs=True,␣
↪processors=-1)
>>> sql_files
{0: Path('cache/06e92da0247c71762d64aed4bcf3cdb2/output_data/
↪06e92da0247c71762d64aed4bcf3cdb2out.sql'),
 1: Path('cache/aee8caf562b3519942ef88f533800dd0/output_data/
↪aee8caf562b3519942ef88f533800dd0out.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,
    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 `OpapqueMaterial` 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],
Surface_Type="Facade",
Outside_Boundary_Condition="Outdoors")

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

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

(continues on next page)

(continued from previous page)

```
    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,
  ```

(continues on next page)

(continued from previous page)

```
    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 = DaySchedule.from_values(
    Name="AlwaysOn", Values=[1] * 24, Type="Fraction", Category="Day"
)
# Always off
sch_d_off = DaySchedule.from_values(
    Name="AlwaysOff", Values=[0] * 24, Type="Fraction", Category="Day"
)
# DHW
sch_d_dhw = DaySchedule.from_values(
    Name="DHW", Values=[0.3] * 24, Type="Fraction", Category="Day"
)
# Internal gains
sch_d_gains = DaySchedule.from_values(
    Name="Gains",
    Values=[0] * 6 + [0.5, 0.6, 0.7, 0.8, 0.9, 1] + [0.7] * 6 + [0.4] * 6,
    Type="Fraction",
    Category="Day",
)
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
# Always on
sch_w_on = WeekSchedule(
    Days=[sch_d_on, sch_d_on, sch_d_on, sch_d_on, sch_d_on, sch_d_on, sch_d_
↪on],
    Category="Week",
    Type="Fraction",
    Name="AlwaysOn",
)
# Always off
sch_w_off = WeekSchedule(
    Days=[
        sch_d_off,
        sch_d_off,
        sch_d_off,
        sch_d_off,
        sch_d_off,
        sch_d_off,
        sch_d_off,
    ],
    Category="Week",
    Type="Fraction",
    Name="AlwaysOff",
)
# DHW
sch_w_dhw = WeekSchedule(
    Days=[
        sch_d_dhw,
        sch_d_dhw,
        sch_d_dhw,
        sch_d_dhw,
        sch_d_dhw,
        sch_d_dhw,
        sch_d_dhw,
    ],
    Category="Week",
    Type="Fraction",
    Name="DHW",
)
# Internal gains
sch_w_gains = WeekSchedule(
    Days=[
        sch_d_gains,
        sch_d_gains,
        sch_d_gains,
        sch_d_gains,
        sch_d_gains,
        sch_d_gains,
        sch_d_gains,
    ],
    Category="Week",
    Type="Fraction",
    Name="Gains",
```

(continues on next page)

(continued from previous page)

```
)
WeekSchedules = [sch_w_on, sch_w_off, sch_w_dhw, sch_w_gains]
```

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

  YearSchedule are created using WeekSchedules defined within *YearSchedulePart* objects. The YearSchedulePart serves to defines the weeks of the year for which the weekly schedule is used. For example, we create YearSchedules from WeekSchedule objects:

```
# YearSchedules using DaySchedule objects
# Always on
sch_y_on = YearSchedule(
    Category="Year",
    Parts=[
        YearSchedulePart(
            FromDay=1, FromMonth=1, ToDay=31, ToMonth=12, Schedule=sch_w_on
        )
    ],
    Type="Fraction",
    Name="AlwaysOn",
)
# Always off
sch_y_off = YearSchedule(
    Category="Year",
    Parts=[
        YearSchedulePart(
            FromDay=1,
            FromMonth=1,
            ToDay=31,
            ToMonth=12,
            Schedule=sch_w_off,
        )
    ],
    Type="Fraction",
    Name="AlwaysOff",
)
# Year ON/OFF
sch_y_on_off = YearSchedule(
    Category="Year",
```

(continues on next page)

(continued from previous page)

```
    Parts=[
        YearSchedulePart(
            FromDay=1, FromMonth=1, ToDay=31, ToMonth=5, Schedule=sch_w_on
        ),
        YearSchedulePart(
            FromDay=1,
            FromMonth=6,
            ToDay=31,
            ToMonth=12,
            Schedule=sch_w_off,
        ),
    ],
    Type="Fraction",
    Name="ON_OFF",
)
# DHW
sch_y_dhw = YearSchedule(
    Category="Year",
    Parts=[
        YearSchedulePart(
            FromDay=1,
            FromMonth=1,
            ToDay=31,
            ToMonth=12,
            Schedule=sch_w_dhw,
        )
    ],
    Type="Fraction",
    Name="DHW",
)
# Internal gains
sch_y_gains = YearSchedule(
    Category="Year",
    Parts=[
        YearSchedulePart(
            FromDay=1,
            FromMonth=1,
            ToDay=31,
            ToMonth=12,
            Schedule=sch_w_gains,
        )
    ],
    Type="Fraction",
    Name="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,
    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="dwh_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",
    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)
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,)
```

(continues on next page)

(continued from previous page)

```
# 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,
    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("file.json")
```

which returns an `UmiTemplateLibrary` object.

### 1.9.1 Combining template libraries

Combine two template libraries like this:

```
from archetypal import UmiTemplateLibrary
lib_a = UmiTemplateLibrary.open("a.json")
lib_b = UmiTemplateLibrary.open("b.json")

lib_c = lib_a + lib_b
```

The resulting *lib_c* will contain all components from both libraries. To avoid duplicates (components that are *equal*), run:

```
lib_c.unique_components()
```

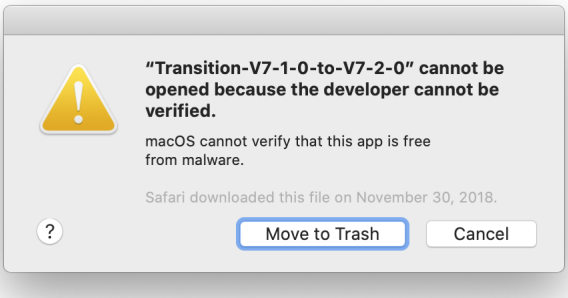
Plot the hierarchy of of an UmiTempalteLIbrary

```
a = UmiTemplateLibrary.open(file)
a.unique_components()
G = a.to_graph()
pos = graphviz_layout(G, prog="dot", args="-s300")
write_dot(G, "G.dot")
fig, ax = plt.subplots(1, 1, figsize=(100,40))
nx.draw(G, pos, with_labels=True, arrows=True, ax=ax)
for group, values in a:
    print("rank = same; " + "; ".join((f'"{v}"' for v in values)) +";")
plt.show()
fig.tight_layout()
fig.savefig("template.pdf")
```

## 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 acheived 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 Settings

—

### 1.12.2 IDF Class

—

### 1.12.3 UMI Template Library

—

### 1.12.4 Template Classes

—

### 1.12.5 Template Helper Classes

Classes that support the *Template Classes* classes above.

—

### 1.12.6 Graph Module

—

### 1.12.7 Schedule Module

—

### 1.12.8 Data Portal

—

### 1.12.9 EnergyDataFrame

**Note:** EnergyDataFrame is now part of its own package energy-pandas.

### 1.12.10 EnergySeries

**Note:** EnergySeries is now part of its own package energy-pandas.

### 1.12.11 Report Data

—

### 1.12.12 Tabular Data

—

### 1.12.13 Utils

| | |
|---|---|
| timeit | Tool for measuring execution time of small code snippets. |

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search