trnslator Documentation

Release 1.1.1

Samuel Letellier-Duchesne
Louis Leroy

Nov 04, 2020

GETTING STARTED

trnslator is a Python package designed with the objective of helping building energy modelers and researchers maintain collections of building archetypes. *trnslator* 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 *trnslator* was created. We built a package able to an EnergyPlus file to TRNBuild models (shout out to TRNSYS users!)

trnslator 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: 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 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*.

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\trnslator>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 trnslator. It must be installed beforehand. Moreover, trnslator 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 *trnslator*, 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 trnslator
```

Hint: If you encounter an issue during the installation of trnslator 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 *trnslator* on a fresh python virtual environment. If you have any trouble with the installation above, try installing trnslator in a new, clean virtual environment using venv or conda. Note that this pacakge was tested with python 3.6:

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

Then you can install trnslator in this freshly created environment:

```
pip install trnslator
```

To use the new environment inside a jupyter notebook, we recommend using the steps described by Angelo Basile:

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

Next time you start a jupyter notebook, you will have the option to choose the *kernel* corresponding to your project, *trnslator* in this case.

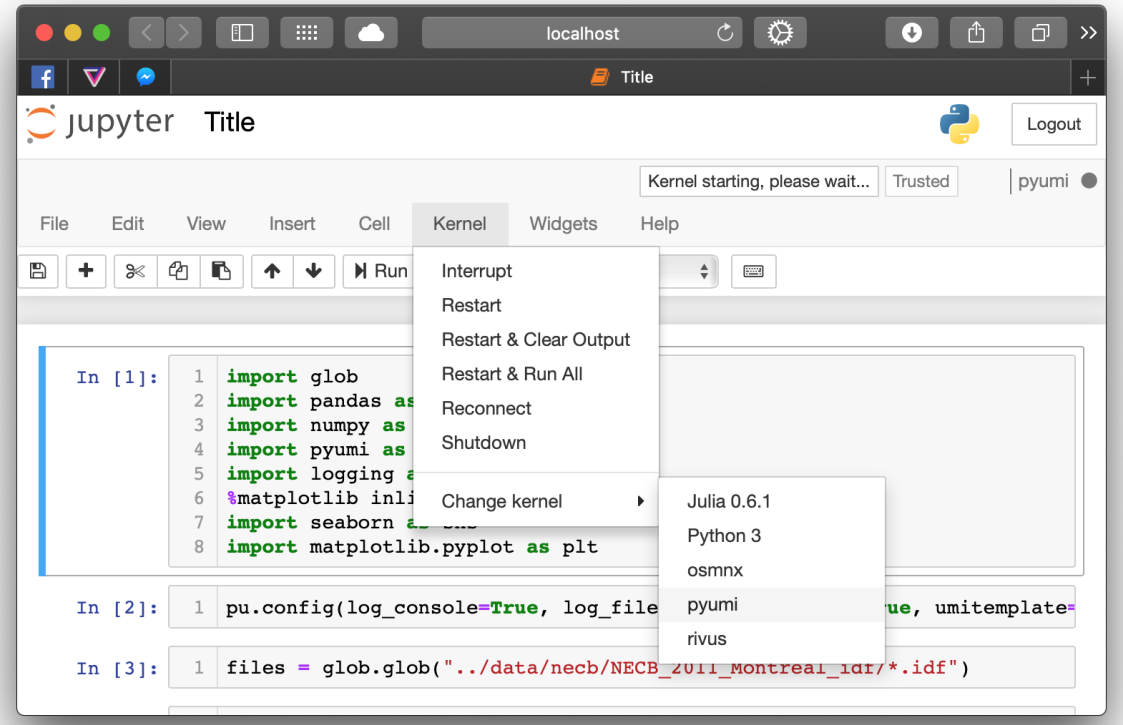


Fig. 1.1: choosing the correct kernel in a jupyter notebook. In the *kernel* menu, select *Change Kernel* and select the appropriate virtual env created earlier (*trnslator* in this case).

1.1.5 Installing using `conda` (Anaconda)

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

Installing with conda is similar to pip. The following workflow creates a new virtual environment (named trnslator) 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 trnslator -f environment.yml
conda activate trnslator
pip install trnslator
```

1.2 For MacOS/Linux users

MacOS or Linux users must install Wine before running trnslator. 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 `trnslator 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 *trnslator convert* command with the `--trnsidf_exe` option.

Example:

```
trnslator 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

Trnslator 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, trnslator 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 *`trnslator.idfclass.run_eplus()`* or the *`trnslator.idfclass.load_idf()`* method 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 `trnslator.idfclass run_eplus()` 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 run_eplus is called a second time (let us say after restarting a Jupyter Notebook kernel), the run_eplus 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 *run_eplus* 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 *`trnslator.utils.config()`* method. The configuration of trnslator 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).

```
import trnslator as tr
tr.config(use_cache=True, log_console=True)
```

1.3.2 Example

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

```
_, idf, results = tr.run_eplus(
    eplus_file=tr.utils.get_eplus_dirs("8-9-0") / "ExampleFiles" / "BasicsFiles/
↪AdultEducationCenter.idf",
    weather_file=tr.utils.get_eplus_dirs("8-9-0") / "WeatherData" / "USA_IL_Chicago-
↪OHare.Intl.AP.725300_TMY3.epw",
    design_day=True,
```

(continues on next page)

(continued from previous page)

```
    return_files=True,
    annual=False,
    return_idf=True,
    expandobjects=True,
    prep_outputs=True,
)
```

Since the file is a version 8.0 IDF file, trnslator is going to transition the file to EnergyPlus 9.2 (or any other version specified with the ep_version parameter) and execute EnergyPlus for the *design_day* only.

The command above yields a list of output files thanks to the *return_files=True* parameter. 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).

```
[None, <trnslator.idfclass.IDF at 0x10fb9f4a8>,
[Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4tbl.csv'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.end'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪AdultEducationCenter.idf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.dxf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.eso'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.mtd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.bnd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.sql'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.mdd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4tbl.htm'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.shd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.expidf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.err'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/eplus_
↪run_AdultEducationCenter.idf_2020_02_27.log'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.mtr'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/sqlite.
↪err'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.audit'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/USA_IL_
↪Chicago-OHare.Intl.AP.725300_TMY3.epw'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.eio'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/d04795a50b4ff172da72fec54c6991e4/
↪d04795a50b4ff172da72fec54c6991e4out.rdd')]]
```

Now, if the command above is modified with *annual=True* and set *design_day=False*, then run_eplus should return

the annual simulation results (which do not exist yet).

```
_, idf, results = tr.run_eplus(
    eplus_file=tr.utils.get_eplus_dirs("8-9-0") / "ExampleFiles" / "BasicsFiles/
↪AdultEducationCenter.idf",
    weather_file=tr.utils.get_eplus_dirs("8-9-0") / "WeatherData" / "USA_IL_Chicago-
↪OHare.Intl.AP.725300_TMY3.epw",
    design_day=False,
    return_files=True,
    annual=True,
    return_idf=True,
    expandobjects=True,
    prep_outputs=True,
)
```

Now, since the original IDF file (the version 8.9 one) has not changed, trnslator 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 run_eplus 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 d04795a50b4ff172da72fec54c6991e4 to 9efc05f6e6cde990685b8ef61e326d94; *these ids may be different on your computer*):

```
[None, <trnslator.idfclass.IDF at 0x1a2c7e0128>,
[Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪AdultEducationCenter.idf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.mdd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.shd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94tbl.htm'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.audit'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.mtr'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.err'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.rdd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.expidf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.eio'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.dxf'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.end'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94tbl.csv'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.eso'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.bnd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.mtd'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/sqlite.
↪err'),
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/
↪9efc05f6e6cde990685b8ef61e326d94out.sql'),
```

(continues on next page)

(continued from previous page)

```
Path('cache/e8f4fb7e50ecaaf2cf2c9d4e4d159605/9efc05f6e6cde990685b8ef61e326d94/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:

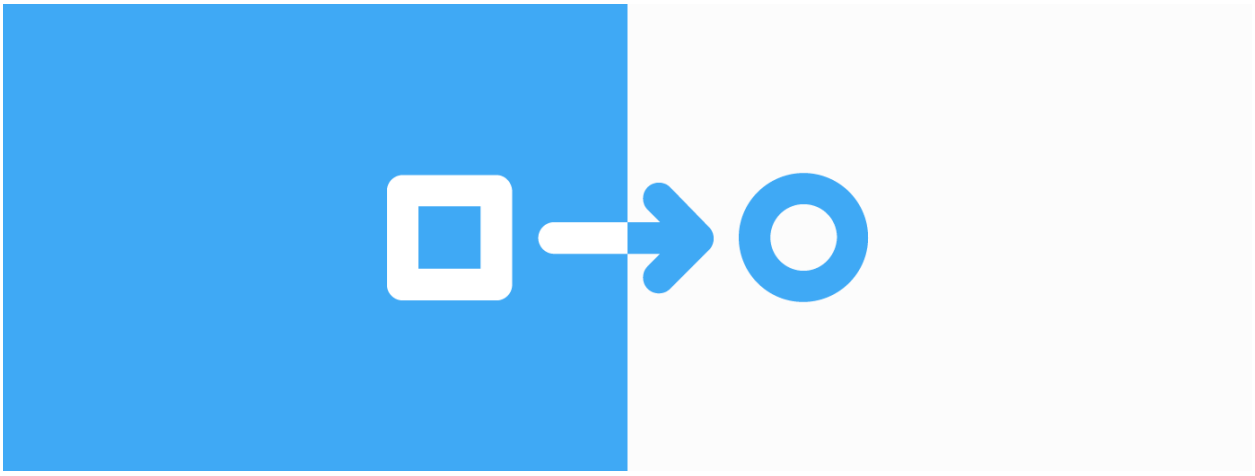
```
Successfully parsed cached idf run in 0.00 seconds
```

Profiling this simple script shows an 8x speedup.

1.4 Converting IDF models

EnergyPlus models can be converted to two different other formats using *trnslator*. TRNSYS users can convert IDF files to TRNBuild/Type56 files.

1.4.1 Converting IDF to BUI



The necessity of translating IDF files (EnergyPlus input files) to BUI files (TRNBuild input files) emerged from the need of modeling building archetypes[1]. Knowing that a lot of different models from different sources (NECB and US-DOE) have already been developed under EnergyPlus, and it can be a tedious task to create a multizone building in a model editor (e.g. TRNBuild), we assume the development of a file translator could be useful for simulationists.

Objectives

The principal objectives of this module was to translate (from IDF to BUI) the geometry of the building, the different schedules used in the model, and the thermal gains.

1. Geometry

The building geometry is kept with all the zoning, the different surfaces (construction and windows) and the thermal properties of the walls. The thermal properties of windows are not from the IDF, but chosen by the user. The user gives a U-value, a SHGC value and Tvis value. Then a window is chosen in the Berkeley Lab library (library used in TRNBuild). For more information, see the *methodology* section please.

[1] Archetype: building model representing a type of building based on its geometry, thermal properties and its usage. Usually used to create urban building model by assigning different archetypes to represent at best the building stock we want to model.

2. Schedules

All schedules from the IDF file are translated. The translator is able to process all schedule types defined by EnergyPlus (see the different schedule types for more information). Only day and week schedules are written in the output BUI file

3. Gains

Internal thermal gains such as "people", "lights" and "equipment" are translated from the IDF file to the BUI file.

4. Conditioning

Heating and cooling demands are translated from the IDF file to the BUI file such as power per floor area (W/m^2) and a temperature setpoint. The temperature set-point is the set-point at the peak time for heating (or cooling).

Methodology

The module is divided in 2 major operations. The first one consist in translating the IDF file from EnergyPlus, to an IDF file proper to an input file for TRNBuild (T3D file), usually created by the TRNSYS plugin "Trnsys3D" in SketchUp. The second operation is the conversion of the IDF file for TRNBuild to a BUI file done with the executable trnsidf.exe (installed by default in the TRNSYS installation folder: *C:TRNSYS18\Building\trnsIDF*)

1. IDF to T3D

The conversion from the IDF EnergyPlus file to the IDF TRNBuild file (called here T3D file) is the important part of the module, which uses the Eppy python package, allowing, with object classes, to find the IDF objects, modify them if necessary and re-transcribe them in the T3D file

2. T3D to BUI

The operation to convert the T3D file to the BUI file is done by running the trnsidf.exe executable with a command line. After this operation, the infiltration rate, internal gains and conditioning systems are written in the "REGIME" section of each zone in the BUI file.

How to convert an IDF file

Converting an IDF file to a BUI file is done using the terminal with a command line. First, open the Command Prompt on Windows or the Terminal on Mac. Note that if you used Anaconda to install python on your machine, you will most likely avoid some issues by using the Anaconda Prompt instead.

Then simply run the following command:

```
trnslator convert [OPTIONS] IDF_FILE WEATHER_FILE OUTPUT_FOLDER
```

1. `IDF_FILE` is the file path of the IDF file to convert. If there are space characters in the path, it should be enclosed in quotation marks.

2. `WEATHER_FILE` is the file path of the weather file to use to run the EnergyPlus simulation. If there are space characters in the path, it should be enclosed in quotation marks.

3. `OUTPUT_FOLDER` is the folder where we want the output folders to be written. If there are space characters in the path, it should enclosed in quotation marks. If output folder path is passed, it must exist. If nothing is passed, the output folder will be the current working directory.

Here is an example. Make sure to replace the last two arguments with the idf file path and the output folder path respectively.

```
trnslator convert "/Users/Documents/NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_
↪PQ_Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/WIP"
```

4. *OPTIONS*: There are different options to the *convert* command. The first 3 manage the requested output files. Users can chose to return a combination of flags

- if `-i` is added, the path to the modified IDF file is returned in the console, and the modified IDF file is returned in the output folder. If `-t` is added, the path to the T3D file (converted from the IDF file) is returned. If `-d` is added, the DCK file (TRNSYS input file) is returned in the output folder, and the path to this DCK file is returned in the console.

```
trnslator convert -i -t -d "/Users/Documents/NECB 2011 - Warehouse.idf" "/
↪Users/Documents/CAN_PQ_Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/
↪WIP"
```

- `--window_lib` is the path of the window library (W74-lib.dat). This library must be in the same format as the Berkeley Lab library used by default in TRNBuild. If nothing is passed, the "W74-lib.dat" file available in the package "ressources" folder will be used.

```
trnslator convert --window_lib "/Users/Documents/W74-lib.dat" "/Users/
↪Documents/NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_PQ_Montreal.
↪Intl.AP.716270_CWEC.epw" "/Users/Documents/WIP"
```

- `--trnsidf_exe` is the path of the trnsidf.exe executable. Usually located in the TRNSYS18 folder under "Building/trnsIDF/trnsidf.exe". If nothing is passed, the following path will be used : "C:TRNSYS18\Building\trnsIDF\trnsidf.exe".

```
trnslator convert --trnsidf_exe "C:TRNSYS18\\Building\\trnsIDF\\trnsidf.exe"
↪"/Users/Documents/NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_PQ_
↪Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/WIP"
```

- `--template` is the path of the .d18 template file (usually in the same directory as the *trnsidf.exe* executable). If nothing is passed, the following path will be used : "C:TRNSYS18\Building\trnsIDF\NewFileTemplate.d18".

```
trnslator convert --template "C:TRNSYS18\\Building\\trnsIDF\\NewFileTemplate.
↪d18" "/Users/Documents/NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_PQ_
↪Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/WIP"
```

- `--log_clear_names` if added, do not print log of "clear_names" (equivalence between old and new names) in the console.

```
trnslator convert --log_clear_names "/Users/Documents/NECB 2011 - Warehouse.
↪idf" "/Users/Documents/CAN_PQ_Montreal.Intl.AP.716270_CWEC.epw" "/Users/
↪Documents/WIP"
```

- `--window` specifies the window properties <u_value (W/m^2-K)> <shgc (-)> <t_vis (-)> <tolerance (-)> <fframe (-)> <uframe (kJ/m^2-K-h)>. If nothing is passed, the following values will be used : 2.2 0.65 0.8 0.05 0.15 8.17

```
trnslator convert --window 2.2 0.65 0.8 0.05 0.15 8.17 "/Users/Documents/
↪NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_PQ_Montreal.Intl.AP.
↪716270_CWEC.epw" "/Users/Documents/WIP"
```

- `--ordered` sorts the idf object names

```
trnslator convert --ordered "/Users/Documents/NECB 2011 - Warehouse.idf" "/
↪Users/Documents/CAN_PQ_Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/
↪WIP"
```

- If `--nonum` is added, do not renumber surfaces in BUI. If `--batchjob` or `-N` is added, does BatchJob Modus when running trnsidf.exe. `--geofloor` must be followed by a float between 0 and 1, and generates GEOSURF values for distributing direct solar radiation where *geo_floor %* is directed to the floor, the rest to walls/windows. If `--refarea` is added, updates floor reference area of airnodes. If `--volume` is added, updates volume of airnodes. If `--capacitance` is added, updates capacitance of airnodes. All those options are used when running trnsidf.exe (converting T3D file to BUI file).

```
trnslator convert --nonum -N --geofloor 0.6 --refarea --volume --capacitance
↪"/Users/Documents/NECB 2011 - Warehouse.idf" "/Users/Documents/CAN_PQ_
↪Montreal.Intl.AP.716270_CWEC.epw" "/Users/Documents/WIP"
```

- `-h` Shows the "help" message

```
trnslator convert -h
```

Equivalence between idf object names when converting a file

Table 1.1: Equivalences

Old names	New names
Building	b
Site:Location	sl
SizingPeriod:DesignDay	spdd
RunPeriod	rp
RunPeriodControl:SpecialDays	rpcs
ScheduleTypeLimits	stl
Schedule:Constant	sc
Schedule:Day:Hourly	sdh
Schedule:Week:Daily	swd
Schedule:Year	sy
Schedule:Compact	sc
Material	m
Material:NoMass	mnm
Material:AirGap	mag
Construction	c
ZoneList	zl
ZoneGroup	zg
Zone	z
BuildingSurface:Detailed	bsd
Shading:Zone:Detailed	szd
InternalMass	im
People	p
Lights	l
ElectricEquipment	ee
GasEquipment	ge
HotWaterEquipment	hwe
SteamEquipment	se
ZoneBaseboard:OutdoorTemperatureControlled	zbot
ZoneInfiltration:DesignFlowRate	zidf
ZoneVentilation:DesignFlowRate	zvdf
ZoneVentilation:WindandStackOpenArea	zvws

continues on next page

Table 1.1 – continued from previous page

Old names	New names
Exterior:Lights	el
Exterior:FuelEquipment	efe
DesignSpecification:OutdoorAir	dsoa
ZoneHVAC:Baseboard:Convective:Electric	zhva
ZoneControl:Thermostat	zct
ThermostatSetpoint:SingleHeating	tssh
ThermostatSetpoint:SingleCooling	tssc
ThermostatSetpoint:DualSetpoint	tsds
AirTerminal:SingleDuct:Uncontrolled	atsd
AirTerminal:SingleDuct:VAV:Reheat	atsd
ZoneHVAC:AirDistributionUnit	zhva
ZoneHVAC:UnitHeater	zhva
ZoneHVAC:HighTemperatureRadiant	zhva
ZoneHVAC:EquipmentList	zhva
Fan:VariableVolume	fvv
Fan:OnOff	foo
Fan:ConstantVolume	fcv
Fan:ZoneExhaust	fze
Coil:Cooling:DX:SingleSpeed	ccdx
Coil:Heating:Electric	che
Coil:Cooling:DX:TwoSpeed	ccdx
Coil:Heating:Fuel	chf
CoilSystem:Cooling:DX	cscd
Controller:MechanicalVentilation	cmv
Controller:OutdoorAir	coa
AirLoopHVAC:Unitary:Furnace:HeatCool	alhv
AirLoopHVAC:ControllerList	alhv
AirLoopHVAC:UnitaryHeatCool	alhv
AirLoopHVAC	alhv
AirLoopHVAC:OutdoorAirSystem:EquipmentList	alhv
AirLoopHVAC:OutdoorAirSystem	alhv
OutdoorAir:Mixer	oam
AirLoopHVAC:ZoneSplitter	alhv
AirLoopHVAC:SupplyPath	alhv
AirLoopHVAC:ReturnPlenum	alhv
AirLoopHVAC:ZoneMixer	alhv
AirLoopHVAC:ReturnPath	alhv
Branch	b
BranchList	bl
NodeList	nl
OutdoorAir:Node	oan
AvailabilityManager:Scheduled	ams
AvailabilityManager:NightCycle	amnc
AvailabilityManagerAssignmentList	amal
SetpointManager:Scheduled	sms
SetpointManager:SingleZone:Reheat	smsz
SetpointManager:MixedAir	smma
WaterHeater:Mixed	whm
WaterUse:Connections	wuc

continues on next page

Table 1.1 – continued from previous page

Old names	New names
WaterUse:Equipment	wue
Curve:Quadratic	cq
Curve:Cubic	cc
Curve:Biquadratic	cb
Controller:OutdoorAir	coa
AirLoopHVAC:ZoneMixer	alhv
OutdoorAir:Node	oan
WaterHeater:Mixed	whm
EnergyManagementSystem:Sensor	emss
EnergyManagementSystem:Actuator	emsa
EnergyManagementSystem:ProgramCallingManager	emsp
EnergyManagementSystem:Program	emsp
EnergyManagementSystem:InternalVariable	emsi
PlantEquipmentList	pel
PlantEquipmentOperation:HeatingLoad	peoh
PlantEquipmentOperationSchemes	peos
PlantLoop	pl
Output:Table:Monthly	otm
Connector:Splitter	cs
Connector:Mixer	cm
ConnectorList	cl
Pipe:Adiabatic	pa
Pump:ConstantSpeed	pcs
LifeCycleCost:Parameters	lccp
LifeCycleCost:NonrecurringCost	lccn
UtilityCost:Tariff	uct
UtilityCost:Variable	ucv

1.5 Reading and running IDF files

trnslator 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 *load_idf()* with the path name. For example:

```
>>> from trnslator import get_eplus_dirs, load_idf
>>> eplus_dir = get_eplus_dirs("9-2-0")  # Getting EnergyPlus install path
>>> eplus_file = eplus_dir / "ExampleFiles" / "BasicsFiles" / "AdultEducationCenter.
↪idf"  # Model path
>>> idf = load_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 = load_idf(eplus_file, weather_file=weather)  # IDF load
```

1.5.2 Editing

Editing IDF files is based on the eppy package. The *IDF* object returned by *load_idf()* 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 *trnslator.idfclass.OutputPrep* class. This class and its methods handle adding predefined or custom outputs to an IDF object. For example, the idf object created above can be modified by adding a basic set of outputs:

```
>>> from trnslator import OutputPrep
>>> OutputPrep(idf=idf, save=True).add_basics()
```

See *OutputPrep* for more details on all possible methods.

1.5.3 Running

Running an EnerguPlus file can be done in two ways. The first way is to call the *trnslator.idfclass.run_eplus()* function with the path of the IDF file and the path of the weather file. The second method is to call the *run_eplus()* method on an *IDF* object that has been previously read. In both cases, users can also specify run options as well as output options. For example, instead of creating an OutputPrep object, one can specify custom outputs in the `trnslator.idfclass.run_eplus.prep_outputs` attribute. These outputs will be appended to the IDF file before the simulation is run. See *run_eplus()* for other parameters to pass to *run_eplus*.

For the same IDF object above, the following:

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

is equivalent to:

```
>>> from trnslator import run_eplus
>>> run_eplus(eplus_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. This can be helpful when *trnslator* is called many times. 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 the various *run_eplus()* options. If caching is activated, then subsequent calls to the *run_eplus()* method will return the cached results.

The caching system is activated by calling the *trnslator.utils.config()* method, 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 trnslator 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 pacakge methods and configure *trnslator* to use caching and to show logs in the console.

```
>>> from path import Path
>>> from trnslator import load_idf, config, run_eplus, settings, parallel_process
>>> import pandas as pd
>>> config(use_cache=True, log_console=True)
```

Then, use

```
>>> from trnslator import load_idf, config, run_eplus, settings
>>> from trnslator 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(
            eplus_file=str(file),
            prep_outputs=True,
            weather_file=str(epw),
            expandobjects=True,
            verbose="v",
            design_day=True,
            output_report="sql_file",
        )
        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'),
```

(continues on next page)

(continued from previous page)

```
 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

trnslator 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

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

```
>>> import trnslator as tr
>>> idf = tr.load_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 TRNSYS* converter. The Schedule module of *trnslator* 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

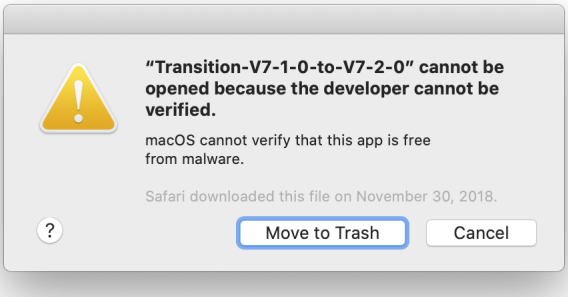
Schedules can be parsed as pandas.Series objects (call the *series* property on a Schedule object) which then exposes useful methods from the pandas package. For convenience, a wrapper for the plotting method is built-in the Schedule class. To plot the full annual schedule (or a specific range), simply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

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

1.8 Troubleshooting

1.8.1 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 trnslator 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.

1.8.2 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/`.

1.9 Command reference

Trnslator 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:

```
trnslator --help
```

1.9.1 trnslator convert

Convert regular IDF file (EnergyPlus) to TRNBuild file (TRNSYS) The output folder path defaults to the working directory. Equivalent to '.'

```
trnslator convert [OPTIONS] IDF_FILE WEATHER_FILE [OUTPUT_FOLDER]
```

Options

-i, --return_idf
Save modified IDF file to output_folder, and return path to the file in the console

-t, --return_t3d
Return T3D file path in the console

-d, --return_dck
Generate dck file and save to output_folder, and return path to the file in the console

--window_lib <window_lib>
Path of the window library (from Berkeley Lab)

--trnsidf_exe <trnsidf_exe>
Path to trnsidf.exe

--template <template>
Path to d18 template file

--log_clear_names
If mentioned (True), DO NOT print log of "clear_names" (equivalence between old and new names) in the console. Default (not mentioned) is False.

--schedule_as_input
If mentioned (False), writes schedules as SCHEDULES in BUI file. Be aware that this option might make crash TRNBuild. Default (not mentioned) is True, and writes the schedules as INPUTS. This option requires the user to link (in the TRNSYS Studio) the csv file containing the schedules with those INPUTS

--ep_version <ep_version>
Specify the EnergyPlus version to use. Default = None

--window <window>
Specify window properties <u_value> <shgc> <t_vis> <tolerance> <fframe> <uframe>. Default = 2.2 0.64 0.8 0.05 0.15 8.17

--ordered
sort idf object names

--nonum
Do not renumber surfaces

-N, --batchjob
BatchJob Modus

--geofloor <geofloor>
Generates GEOSURF values for distributing; direct solar radiation where *geo_floor %* is directed to the floor, the rest; to walls/windows. Default = 0.6

--refarea
Updates floor reference area of airnodes

--volume
Updates volume of airnodes

--capacitance
Updates capacitance of airnodes

Arguments

IDF_FILE
Required argument

WEATHER_FILE
Required argument

OUTPUT_FOLDER
Optional argument

1.9.2 trnslator transition

Upgrade an IDF file to a newer version

```
trnslator transition [OPTIONS] IDF...
```

Options

-v, --version <to_version>
EnergyPlus version to upgrade to - e.g., '9-2-0'

-p, --parallel <cores>
Specify number of cores to run in parallel

Arguments

IDF
Required argument(s)

1.10 Modules

1.10.1 IDF Class

load_idf	Returns a parsed IDF object from file.
IDF	Wrapper over the geomeppy.IDF class and subsequently the eppy.modeleditor.IDF class
run_eplus	Run an EnergyPlus file using the EnergyPlus executable.
OutputPrep	Handles preparation of EnergyPlus outputs.

trnslator.idfclass.load_idf

trnslator.idfclass.**load_idf**(*eplus_file*, *idd_filename=None*, *output_folder=None*, *include=None*, *weather_file=None*, *ep_version=None*)

Returns a parsed IDF object from file. If *trnslator.settings.use_cache* is true, then the idf object is loaded from cache.

Parameters

- **eplus_file** (*str*) – Either the absolute or relative path to the idf file.
- **idd_filename** (*str, optional*) – Either the absolute or relative path to the EnergyPlus IDD file. If None, the function tries to find it at the default EnergyPlus install location.
- **output_folder** (*Path, optional*) – Either the absolute or relative path of the output folder. Specify if the cache location is different than trnslator.settings.cache_folder.
- **include** (*str, optional*) – List input files that need to be copied to the simulation directory. Those can be, for example, schedule files read by the idf file. If a string is provided, it should be in a glob form (see pathlib.Path.glob).
- **weather_file** – Either the absolute or relative path to the weather epw file.
- **ep_version** (*str, optional*) – EnergyPlus version number to use, eg.: "9-2-0". Defaults to *settings.ep_version* .

Returns The IDF object.

Return type *IDF*

trnslator.idfclass.IDF

class trnslator.idfclass.**IDF**(**args*, ***kwargs*)

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

Parameters

- ***args** –
- ****kwargs** –

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

Set the path to the EnergyPlus IDD for the version of EnergyPlus which is to be used by eppy.

Parameters

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

property area_conditioned

Returns the total conditioned area of a building (taking into account zone multipliers

property partition_ratio

The number of lineal meters of partitions (Floor to ceiling) present in average in the building floor plan by m2.

wwr(*azimuth_threshold=10*, *round_to=None*)

Returns the Window-to-Wall Ratio by major orientation for the IDF model. 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={}*)

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

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

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={}*)

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

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

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

run_eplus(***kwargs*)

wrapper around the *trnslator.idfclass.run_eplus()* method.

If weather file is defined in the IDF object, then this field is optional. By default, will load the sql in self.sql.

Parameters **kwargs** –

Returns The output report or the sql file loaded as a dict of DataFrames.

add_object(*ep_object*, *save=True*, ***kwargs*)

Add a new object to an idf file. The function will test if the object exists to prevent duplicates. By default, the idf with the new object is saved to disk (save=True)

Parameters

- **ep_object** (*str*) – the object name to add, eg. 'OUTPUT:METER' (Must be in all_caps).

- **save** (*bool*) – Save the IDF as a text file with the current idfname of the IDF.
- ****kwargs** – keyword arguments to pass to other functions.

Returns the object

Return type EpBunch

get_schedule_type_limits_data_by_name(*schedule_limit_name*)
Returns the data for a particular 'ScheduleTypeLimits' object

Parameters **schedule_limit_name** –

get_schedule_epbunch(*name*, *sch_type=None*)
Returns the epbunch of a particular schedule name. If the schedule type is know, retreives it quicker.

Parameters

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

get_all_schedules(*yearly_only=False*)
Returns all schedule ep_objects in a dict with their name as a key

Parameters **yearly_only** (*bool*) – If True, return only yearly schedules

Returns

the schedules with their name as a key

Return type (dict of eppy.bunch_subclass.EpBunch)

get_used_schedules(*yearly_only=False*)
Returns all used schedules

Parameters **yearly_only** (*bool*) – If True, return only yearly schedules

Returns the schedules names

Return type (list)

property day_of_week_for_start_day
Get day of week for start day for the first found RUNPERIOD

building_name(*use_idfname=False*)

Parameters **use_idfname** –

rename(*objkey*, *objname*, *newname*)
rename all the references to this objname

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

Parameters

- **objkey** (*EpBunch*) – 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 IDF objects renameds

Return type theobject (EpBunch)

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', 'CondenserLoop'],

'Compliance Objects': ['Compliance:Building'], 'Controllers': ['Controller:WaterCoil',

'Controller:OutdoorAir', 'Controller:MechanicalVentilation', 'AirLoopHVAC:ControllerList'],

...}

Returns

Return type dict

classmethod getiddname()
Get the name of the current IDD used by eppy.

Returns

Return type str

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.

newidfobject(*key: str*, *aname: str = ''*, ***kwargs: Any*) → geomeppy.patches.EpBunch
Add a new idfobject to the model.

If you don't specify a value for a field, the default value will be set.

For example

```
newidfobject("CONSTRUCTION")
newidfobject("CONSTRUCTION",
    Name='Interior Ceiling_class',
    Outside_Layer='LW Concrete',
    Layer_2='soundmat')
```

Parameters

- **key** – The type of IDF object. This must be in ALL_CAPS.
- **aname** – 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 EpBunch object.

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.

read()
Read the IDF file and the IDD file.

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

Populates the following data structures:

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

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

removeidfobject(*idfobject*)
Remove an IDF object from the IDF.

Parameters **idfobject** (*EpBunch object*) – The IDF object to remove.

rotate(*angle: float*, *anchor: Optional[Union[geomeppy.geom.vectors.Vector2D, geomeppy.geom.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,
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.

save(*filename=None*, *lineendings='default'*, *encoding='latin-1'*)
Save the IDF as a text file with the optional filename passed, or with the current idfname of the IDF.

Parameters

- **filename** (*str, optional*) – Filepath to save the file. If None then use the IDF.idfname parameter. Also accepts a file handle.
- **lineendings** (*str, optional*) – 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, optional*) – Encoding to use for the saved file. The default is 'latin-1' which is compatible with the EnergyPlus IDFEditor.

saveas(*filename*, *lineendings='default'*, *encoding='latin-1'*)
Save the IDF as a text file with the filename passed.

Parameters

- **filename** (*str*) – Filepath to to set the idfname attribute to and save the file as.

- **lineendings** (*str, optional*) – 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, optional*) – Encoding to use for the saved file. The default is 'latin-1' which is compatible with the EnergyPlus IDFEditor.

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, optional*) – 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, optional*) – Encoding to use for the saved file. The default is 'latin-1' which is compatible with the EnergyPlus IDFEditor.

scale (*factor: float*, *anchor: Optional[Union[geomeppy.geom.vectors.Vector2D, geomeppy.geom.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'.

set_wwr (*wwr: Optional[float] = 0.2*, *construction: Optional[str] = None*, *force: Optional[bool] = False*, *wwr_map: Optional[dict] = {}*, *orientation: Optional[str] = None*) → None
Add strip windows to all external walls.

Different WWR can be applied to specific wall orientations using the *wwr_map* keyword arg. This map is a dict of wwr values, keyed by *wall.azimuth*, which overrides the default passed as *wwr*.

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** – Window to wall ratio in the range 0.0 to 1.0.
- **construction** – Name of a window construction.
- **force** – True to remove all subsurfaces before setting the WWR.
- **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.

classmethod setidd (*iddinfo*, *iddindex*, *block*, *idd_version*)
Set the IDD to be used by eppy.

Parameters

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

to_obj(*fname: Optional[str] = None*, *mtllib: 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.
- **mtllib** – The name of a .mtl file to be referenced from the .obj file. If None, we use default.mtl.

translate(*vector: geomeppy.geom.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.

trnslator.idfclass.run_eplus

trnslator.idfclass.**run_eplus**(*eplus_file*, *weather_file*, *output_directory=None*, *ep_version=None*, *output_report=None*, *prep_outputs=False*, *simulname=None*, *keep_data=True*, *annual=False*, *design_day=False*, *epmacro=False*, *expandobjects=True*, *readvars=False*, *output_prefix=None*, *output_suffix=None*, *version=None*, *verbose='v'*, *keep_data_err=False*, *include=None*, *process_files=False*, *custom_processes=None*, *return_idf=False*, *return_files=False*, ***kwargs*)
Run an EnergyPlus file using the EnergyPlus executable.

Specify run options: Run options are specified in the same way as the E+ command line interface: annual, design_day, epmacro, expandobjects, etc. are all supported.

Specify outputs: Optionally define the desired outputs by specifying the `prep_outputs` attribute.

With the `prep_outputs` attribute, specify additional outputs objects to append to the energy plus file. If True is specified, a selection of useful options will be append by default (see: *OutputPrep* for more details).

Parameters

- **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. Will default to the settings.cache_folder.
- **ep_version** (*str, optional*) – EnergyPlus 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.
- **simulname** (*str*) – The name of the simulation. (Todo: Currently not implemented).

- **keep_data** (*bool*) – If True, files created by EnergyPlus are saved to the output_directory.
- **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 occurred is 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*) – It True, all files paths created by the EnergyPlus run are returned.

Returns

a 1-tuple or a 2-tuple

- dict: dict of [(title, table), ...]
- IDF: The IDF object. Only provided if return_idf is True.

Return type 2-tuple

Raises **EnergyPlusProcessError.** –

trnslator.idfclass.OutputPrep

class trnslator.idfclass.**OutputPrep**(*idf*, *save=True*)

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

For example:

```
>>> OutputPrep(idf=idf_obj).add_output_control().add_umi_ouputs().add_profile_
↪gas_elect_ouputs()
```

Initialize an OutputPrep object.

Parameters

- **idf** (*IDF*) – the IDF object for wich this OutputPrep object is created.
- **save** (*bool*) – weather to save or not changes after adding outputs to the IDF file.

add_custom(*outputs*)

Add custom-defined outputs as a list of objects.

Examples

```
>>> outputs = [
>>>         {
>>>             "ep_object": "OUTPUT:METER",
>>>             "kwargs": dict(
>>>                 Key_Name="Electricity:Facility",
>>>                 Reporting_Frequency="hourly",
>>>                 save=True,
>>>             ),
>>>         },
>>>     ]
>>> OutputPrep().add_custom(outputs)
```

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

add_basics()

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

add_schedules()

Adds Schedules object

add_summary_report(*summary='AllSummary'*)

Adds the Output:Table:SummaryReports object.

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

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.

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

add_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

1.10.2 Schedule Module

Schedule	An object designed to handle any EnergyPlus schedule object

trnslator.schedule.Schedule

class trnslator.schedule.**Schedule**(*Name=None*, *idf=None*, *start_day_of_the_week=0*, *strict=False*, *base_year=2018*, *schType=None*, *schTypeLimitsName=None*, *values=None*, ***kwargs*)
An object designed to handle any EnergyPlus schedule object

Parameters

- **Name** (*str*) – The schedule name in the idf file
- **idf** (IDF) – IDF object
- **start_day_of_the_week** (*int*) – 0-based day of week (Monday=0)
- **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.
- **base_year** (*int*) – The base year of the schedule. Defaults to 2018 since the first day of that year is a Monday.
- **schType** (*str*) – The EnergyPlus schedule type. eg.: "Schedule:Year"
- **schTypeLimitsName** –
- **values** –
- ****kwargs** –

classmethod from_values(*Name*, *values*, ***kwargs*)

Parameters

- **Name** –
- **values** –
- ****kwargs** –

classmethod constant_schedule(*hourly_value=1*, *Name='AlwaysOn'*, *idf=None*, ***kwargs*)
Create a schedule with a constant value for the whole yetr. Defaults to a schedule with a value of 1, named 'AlwaysOn'.

Parameters

- **hourly_value** (*float, optional*) – The value for the constant schedule. Defaults to 1.
- **Name** (*str, optional*) – The name of the schedule. Defaults to Always On.
- **idf** –
- ****kwargs** –

property all_values
returns the values array

property series
Returns the schedule values as a pd.Series object with a DateTimeIndex

get_schedule_type_limits_name(*sch_type=None*)
Return the Schedule Type Limits name associated to this schedule

Parameters **sch_type** –

get_schedule_type_limits_data(*name=None*)
Returns Schedule Type Limits data from schedule name

Parameters **name** –

get_schedule_type(*name=None*)
Return the schedule type, eg.: "Schedule:Year"

Parameters **name** –

start_date()
The start date of the schedule. Satisfies *startDayOfTheWeek*

plot(*slice=None*, ***kwargs*)
Plot the schedule. Implements the .loc accessor on the series object.

Examples

```
>>> s = Schedule(
>>>          Name="NECB-A-Thermostat Setpoint-Heating",
>>>          idf=idf_object)
>>>      )
>>> s.plot(slice=("2018/01/02", "2018/01/03"), drawstyle="steps-post")
>>> plt.show()
```

Parameters

- **slice** (*tuple*) – define a 2-tuple object the will be passed to `pandas.IndexSlice` as a range.
- ****kwargs** (*dict*) – keyword arguments passed to `pandas.Series.plot()`.

get_interval_day_ep_schedule_values(*epbunch*)
Schedule:Day:Interval

Parameters **epbunch** (*EpBunch*) – The schedule EpBunch object.

get_hourly_day_ep_schedule_values(*epbunch*)
Schedule:Day:Hourly

Parameters **epbunch** (*EpBunch*) – The schedule EpBunch object.

get_compact_weekly_ep_schedule_values(*epbunch*, *start_date=None*, *index=None*)
schedule:week:compact

Parameters

- **epbunch** (*EpBunch*) – the name of the schedule
- **start_date** –
- **index** –

get_daily_weekly_ep_schedule_values(*epbunch*)
schedule:week:daily

Parameters **epbunch** (*EpBunch*) – The schedule EpBunch object.

get_list_day_ep_schedule_values(*epbunch*)
schedule:day:list

Parameters **epbunch** (*EpBunch*) – The schedule epbunch object.

get_constant_ep_schedule_values(*epbunch*)
schedule:constant

Parameters **epbunch** (*EpBunch*) – The schedule epbunch object.

get_file_ep_schedule_values(*epbunch*)
schedule:file

Parameters **epbunch** (*EpBunch*) – The schedule epbunch object.

get_compact_ep_schedule_values(*epbunch*)
schedule:compact

Parameters **epbunch** (*EpBunch*) – The schedule epbunch object.

static invalidate_condition(*series*)

Parameters **series** –

get_yearly_ep_schedule_values(*epbunch*)
schedule:year

Parameters **epbunch** (*EpBunch*) – the schedule epbunch.

get_schedule_values(*sched_epbunch*, *start_date=None*, *index=None*)
Main function that returns the schedule values

Parameters

- **sched_epbunch** (*EpBunch*) – the schedule epbunch object

- **start_date** –
- **index** –

to_year_week_day (*values=None*, *idf=None*)
convert a Schedule Class to the 'Schedule:Year', 'Schedule:Week:Daily' and 'Schedule:Day:Hourly' representation

Parameters

- **values** –
- **idf** –

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

field_set (*field*, *slicer_=None*)
helper function to return the proper slicer depending on the field_set value.

Available values are: Weekdays, Weekends, Holidays, Alldays, SummerDesignDay, WinterDesignDay, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, CustomDay1, CustomDay2, AllOtherDays

Parameters

- **field** (*str*) – The EnergyPlus field set value.
- **slicer** –

Returns Returns the appropriate indexer for the series.

Return type (indexer-like)

get_sdow (*start_day_of_week*)
Returns the start day of the week

Parameters **start_day_of_week** –

special_day (*field*, *slicer_*)
try to get the RunPeriodControl:SpecialDays for the corresponding Day Type

Parameters

- **field** –
- **slicer** –

design_day (*field*, *slicer_*)

Parameters

- **field** –
- **slicer** –

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*)

1.10.3 EnergyDataFrame

EnergyDataFrame.set_unit	
EnergyDataFrame.discretize_tsam	uses tsam
plot_energydataframe_map	

trnslator.energydataframe.EnergyDataFrame.set_unit

EnergyDataFrame.**set_unit**(*from_unit*, *inplace*)

trnslator.energydataframe.EnergyDataFrame.discretize_tsam

EnergyDataFrame.**discretize_tsam**(*resolution=None*, *noTypicalPeriods=10*, *hoursPerPeriod=24*, *clusterMethod='hierarchical'*, *evalSumPeriods=False*, *sortValues=False*, *sameMean=False*, *rescaleClusterPeriods=True*, *weightDict=None*, *extremePeriodMethod='None'*, *solver='glpk'*, *roundOutput=None*, *addPeakMin=None*, *addPeakMax=None*, *addMeanMin=None*, *addMeanMax=None*)

uses tsam

trnslator.energydataframe.plot_energydataframe_map

trnslator.energydataframe.**plot_energydataframe_map**(*data*, *periodlength=24*, *subplots=False*, *vmin=None*, *vmax=None*, *axis_off=True*, *cmap='RdBu'*, *fig_height=None*, *fig_width=6*, *show=True*, *view_angle=- 60*, *save=False*, *close=False*, *dpi=300*, *file_format='png'*, *color=None*, *ax=None*, *filename='untitled'*, *extent='tight'*, *sharex=True*, *sharey=True*, *layout=None*, *layout_type='vertical'*, ***kwargs*)

1.10.4 EnergySeries

EnergySeries.from_sqlite	Create a.
EnergySeries.unit_conversion	returns the multiplier to convert units
EnergySeries.concurrent_sort	**param ascending**
EnergySeries.normalize	Returns a normalized EnergySeries
EnergySeries.ldc_source	Returns the Load Duration Curve from the source side of theoretical Heat Pumps
EnergySeries.source_side	Returns the Source Side EnergySeries given a Seasonal COP.
EnergySeries.discretize_tsam	uses tsam
EnergySeries.discretize	Retruns a discretized pandas.Series
EnergySeries.plot3d	**param energy_series**
EnergySeries.plot2d	**param data**
EnergySeries.p_max	
EnergySeries.p_max	
EnergySeries.monthly	
EnergySeries.capacity_factor	
EnergySeries.bin_edges	
EnergySeries.time_at_min	
EnergySeries.bin_scaling_factors	
EnergySeries.duration_scaling_factor	
EnergySeries.ldc	
EnergySeries.nseries	
save_and_show	Save a figure to disk and show it, as specified.
plot_energyseries	**param energy_series**
plot_energyseries_map	**param data**

trnslator.energyseries.EnergySeries.from_sqlite

classmethod EnergySeries.**from_sqlite**(*df*, *name=None*, *base_year=2018*, *units=None*, *normalize=False*, *sort_values=False*, *ascending=False*, *concurrent_sort=False*, *to_units=None*, *agg_func='sum'*)

Create a.

Parameters

- **df** (*DataFrame*) –
- **name** –
- **base_year** –

- **units** –
- **normalize** –
- **sort_values** –
- **ascending** –
- **concurrent_sort** –
- **to_units** –
- **agg_func** (*callable*) – The aggregation function to use in the case that multiple values have the same index value. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply. For a DataFrame, can pass a dict, if the keys are DataFrame column names.

  Accepted Combinations are:

  - string function name
  - function
  - list of functions
  - dict of column names -> functions (or list of functions)

trnslator.energyseries.EnergySeries.unit_conversion

EnergySeries.**unit_conversion**(*to_units=None*, *inplace=False*)
returns the multiplier to convert units

Parameters

- **to_units** (*pint.Unit*) –
- **inplace** –

trnslator.energyseries.EnergySeries.concurrent_sort

EnergySeries.**concurrent_sort**(*ascending=False*, *inplace=False*, *level=0*)

Parameters

- **ascending** –
- **inplace** –
- **level** –

trnslator.energyseries.EnergySeries.normalize

EnergySeries.**normalize**(*feature_range=(0, 1)*, *inplace=False*)
Returns a normalized EnergySeries

Parameters

- **feature_range** –
- **inplace** –

trnslator.energyseries.EnergySeries.ldc_source

EnergySeries.**ldc_source**(*SCOPH=4*, *SCOPC=4*)

Returns the Load Duration Curve from the source side of theoretical Heat Pumps

Parameters

- **SCOPH** – Seasonal COP in Heating
- **SCOPC** – Seasonal COP in Cooling

Returns (EnergySeries) Load Duration Curve

trnslator.energyseries.EnergySeries.source_side

EnergySeries.**source_side**(*SCOPH=None*, *SCOPC=None*)

Returns the Source Side EnergySeries given a Seasonal COP. Negative values are considered like Cooling Demand.

Parameters

- **SCOPH** – Seasonal COP in Heating
- **SCOPC** – Seasonal COP in Cooling

Returns (EnergySeries) Load Duration Curve

trnslator.energyseries.EnergySeries.discretize_tsam

EnergySeries.**discretize_tsam**(*resolution=None*, *noTypicalPeriods=10*, *hoursPerPeriod=24*, *clusterMethod='hierarchical'*, *evalSumPeriods=False*, *sortValues=False*, *sameMean=False*, *rescaleClusterPeriods=True*, *weightDict=None*, *extremePeriodMethod='None'*, *solver='glpk'*, *roundOutput=None*, *addPeakMin=None*, *addPeakMax=None*, *addMeanMin=None*, *addMeanMax=None*)

uses tsam

Parameters

- **resolution** –
- **noTypicalPeriods** –
- **hoursPerPeriod** –
- **clusterMethod** –
- **evalSumPeriods** –
- **sortValues** –
- **sameMean** –
- **rescaleClusterPeriods** –
- **weightDict** –
- **extremePeriodMethod** –
- **solver** –
- **roundOutput** –
- **addPeakMin** –

- **addPeakMax** –
- **addMeanMin** –
- **addMeanMax** –

trnslator.energyseries.EnergySeries.discretize

EnergySeries.**discretize**(*n_bins=3*, *inplace=False*)

Retruns a discretized pandas.Series

Parameters

- **n_bins** (*int*) – Number of bins or steps to discretize the function
- **inplace** (*bool*) – if True, perform operation in-place

trnslator.energyseries.EnergySeries.plot3d

EnergySeries.**plot3d**(**args*, ***kwargs*)

Parameters

- **energy_series** (*EnergySeries*) –
- **kind** (*str*) –
- **axis_off** (*bool*) –
- **cmap** –
- **fig_height** (*float*) –
- **fig_width** (*float*) –
- **show** (*bool*) –
- **view_angle** (*float*) –
- **save** (*bool*) –
- **close** (*bool*) –
- **dpi** (*int*) –
- **file_format** (*str*) –
- **color** (*str*) –
- **axes** –
- **vmin** (*float*) –
- **vmax** (*float*) –
- **filename** (*str*) –
- **timeStepsPerPeriod** (*int*) – The number of discrete timesteps which describe one period.
- ****kwargs** –

trnslator.energyseries.EnergySeries.plot2d

EnergySeries.**plot2d**(**args*, ***kwargs*)

Parameters

- **data** (*EnergySeries or EnergyDataFrame*) –
- **periodlength** –
- **subplots** –
- **vmin** –
- **vmax** –
- **axis_off** –
- **cmap** –
- **fig_height** –
- **fig_width** –
- **show** –
- **view_angle** –
- **save** –
- **close** –
- **dpi** –
- **file_format** –
- **color** –
- **ax** –
- **filename** –
- **extent** –
- **sharex** –
- **sharey** –
- **layout** –
- **layout_type** –
- ****kwargs** –

trnslator.energyseries.EnergySeries.p_max

property EnergySeries.**p_max**

trnslator.energyseries.EnergySeries.monthly

property EnergySeries.**monthly**

trnslator.energyseries.EnergySeries.capacity_factor

property EnergySeries.**capacity_factor**

trnslator.energyseries.EnergySeries.bin_edges

property EnergySeries.**bin_edges**

trnslator.energyseries.EnergySeries.time_at_min

property EnergySeries.**time_at_min**

trnslator.energyseries.EnergySeries.bin_scaling_factors

property EnergySeries.**bin_scaling_factors**

trnslator.energyseries.EnergySeries.duration_scaling_factor

property EnergySeries.**duration_scaling_factor**

trnslator.energyseries.EnergySeries.ldc

property EnergySeries.**ldc**

trnslator.energyseries.EnergySeries.nseries

property EnergySeries.**nseries**

trnslator.energyseries.save_and_show

trnslator.energyseries.**save_and_show**(*fig*, *ax*, *save*, *show*, *close*, *filename*, *file_format*, *dpi*, *axis_off*, *extent*)

Save a figure to disk and show it, as specified.

Parameters

- **fig** (*matplotlib.figure.Figure*) – the figure
- **ax** (*matplotlib.axes.Axes or list(matplotlib.axes.Axes)*) – the axes
- **save** (*bool*) – whether to save the figure to disk or not
- **show** (*bool*) – whether to display the figure or not
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **filename** (*string*) – the name of the file to save
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')

- **dpi** (*int*) – the resolution of the image file if saving (Dots per inch)
- **axis_off** (*bool*) – if True matplotlib axis was turned off by plot_graph so constrain the saved figure's extent to the interior of the axis
- **extent** –

Returns (tuple) fig, ax

trnslator.energyseries.plot_energyseries

trnslator.energyseries.**plot_energyseries**(*energy_series*, *kind='polygon'*, *axis_off=True*, *cmap=None*, *fig_height=None*, *fig_width=6*, *show=True*, *view_angle=- 60*, *save=False*, *close=False*, *dpi=300*, *file_format='png'*, *color=None*, *axes=None*, *vmin=None*, *vmax=None*, *filename=None*, *timeStepsPerPeriod=24*, ***kwargs*)

Parameters

- **energy_series** (*EnergySeries*) –
- **kind** (*str*) –
- **axis_off** (*bool*) –
- **cmap** –
- **fig_height** (*float*) –
- **fig_width** (*float*) –
- **show** (*bool*) –
- **view_angle** (*float*) –
- **save** (*bool*) –
- **close** (*bool*) –
- **dpi** (*int*) –
- **file_format** (*str*) –
- **color** (*str*) –
- **axes** –
- **vmin** (*float*) –
- **vmax** (*float*) –
- **filename** (*str*) –
- **timeStepsPerPeriod** (*int*) – The number of discrete timesteps which describe one period.
- ****kwargs** –

trnslator.energyseries.plot_energyseries_map

trnslator.energyseries.**plot_energyseries_map**(*data*, *periodlength=24*, *subplots=False*, *vmin=None*, *vmax=None*, *axis_off=True*, *cmap='RdBu'*, *fig_height=None*, *fig_width=6*, *show=True*, *view_angle=-60*, *save=False*, *close=False*, *dpi=300*, *file_format='png'*, *color=None*, *ax=None*, *filename='untitled'*, *extent='tight'*, *sharex=False*, *sharey=False*, *layout=None*, *layout_type='vertical'*, ***kwargs*)

Parameters

- **data** (*EnergySeries or EnergyDataFrame*) –
- **periodlength** –
- **subplots** –
- **vmin** –
- **vmax** –
- **axis_off** –
- **cmap** –
- **fig_height** –
- **fig_width** –
- **show** –
- **view_angle** –
- **save** –
- **close** –
- **dpi** –
- **file_format** –
- **color** –
- **ax** –
- **filename** –
- **extent** –
- **sharex** –
- **sharey** –
- **layout** –
- **layout_type** –
- ****kwargs** –

1.10.5 Report Data

ReportData.from_sql_dict	
ReportData.from_sqlite	Reads an EnergyPlus eplusout.sql file and returns a ReportData which is a subclass of DataFrame.
ReportData.heating_load	Returns the aggragated 'Heating:Electricity', 'Heating:Gas' and 'Heating:DistrictHeating' of each archetype
ReportData.filter_report_data	filter RaportData using specific keywords.
ReportData.sorted_values	Returns sorted values by filtering key_value and name

trnslator.reportdata.ReportData.from_sql_dict

classmethod ReportData.**from_sql_dict**(*sql_dict*)

trnslator.reportdata.ReportData.from_sqlite

classmethod ReportData.**from_sqlite**(*sqlite_file*, *table_name='WaterSystems:EnergyTransfer'*, *warmup_flag=0*, *environment_type=3*)

Reads an EnergyPlus eplusout.sql file and returns a ReportData which is a subclass of DataFrame.

Parameters

- **environment_type** (*str*) – 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.
- **sqlite_file** (*str*) –

Returns The ReportData object.

Return type (ReportData)

trnslator.reportdata.ReportData.heating_load

ReportData.**heating_load**(*normalize=False*, *sort=False*, *ascending=False*, *concurrent_sort=False*)

Returns the aggragated 'Heating:Electricity', 'Heating:Gas' and 'Heating:DistrictHeating' of each archetype

Parameters

- **normalize** (*bool*) – if True, returns a normalize Series. Normalization is done with respect to each Archetype
- **sort** (*bool*) – if True, sorts the values. Usefull when a load duration curve is needed.
- **ascending** (*bool*) – if True, sorts value in ascending order. If a Load Duration Curve is needed, use ascending=False.

Returns

the Value series of the Heating Load with a Archetype, TimeIndex as MultiIndex.

Return type EnergySeries

trnslator.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 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*) –
- **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

trnslator.reportdata.ReportData.sorted_values

ReportData.**sorted_values**(*key_value=None*, *name=None*, *by='TimeIndex'*, *ascending=True*)

Returns sorted values by filtering key_value and name

Parameters

- **self** – The ReporatData DataFrame
- **key_value** (*str*) – key_value column filter
- **name** (*str*) – name column filter
- **by** (*str*) – sorting by this column name
- **ascending** (*bool*) –

Returns ReportData

1.10.6 IDF to BUI module

convert_idf_to_trnbuild	Convert regular IDF file (EnergyPlus) to TRNBuild file (TRNSYS)
get_idf_objects	Gets idf objects
clear_name_idf_objects	Clean names of IDF objects.
zone_origin	Return coordinates of a zone
closest_coords	Find closest coordinates to given ones
parse_window_lib	Function that parse window library from Berkeley Lab in two parts.
choose_window	Return window object from TRNBuild library
trnbuild_idf	This program sorts and renumbers the IDF file and writes a B18 file based on the geometric information of the IDF file and the template D18 file.

trnslator.trnsys.convert_idf_to_trnbuild

trnslator.trnsys.**convert_idf_to_trnbuild**(*idf_file*, *weather_file*, *window_lib=None*, *return_idf=False*, *return_b18=True*, *return_t3d=False*, *return_dck=False*, *output_folder=None*, *trnsidf_exe=None*, *template=None*, *log_clear_names=False*, *schedule_as_input=True*, ***kwargs*)

Convert regular IDF file (EnergyPlus) to TRNBuild file (TRNSYS)

There are three optional outputs:

- **the path to the modified IDF with the new names, coordinates, etc. of** the IDF objects. It is an input file for EnergyPlus (.idf)
- the path to the TRNBuild file (.b18)
- the path to the TRNBuild input file (.idf)
- the path to the TRNSYS dck file (.dck)

Example

```
>>> # Exemple of setting kwargs to be unwrapped in the function
>>> kwargs_dict = {'u_value': 2.5, 'shgc': 0.6, 't_vis': 0.78,
>>>                'tolerance': 0.05, "fframe": 0.0, "uframe": 0.5, 'ordered':␣
↪True}
>>> # Exemple how to call the function
>>> idf_file = "/file.idf"
>>> window_filepath = "/W74-lib.dat"
>>> convert_idf_to_trnbuild(idf_file=idf_file, weather_file=weather_file,
>>>                         window_lib=window_filepath,
>>>                         **kwargs_dict)
```

Parameters

- **idf_file** (*str*) – path to the idf file to convert
- **weather_file** (*str*) – To run EnergyPlus simulation and be able to get some values (e.g. internal gain, infiltration, etc.)

- **window_lib** (*str*) – File path of the window library (from Berkeley Lab)
- **return_idf** (*bool, optional*) – If True, also return the path to the modified IDF with the new names, coordinates, etc. of the IDF objects. It is an input file for EnergyPlus (.idf)
- **return_b18** (*bool, optional*) – If True, also return the path to the TRNBuild file (.b18).
- **return_t3d** (*bool, optional*) – If True, also return the path to the
- **return_dck** (*bool, optional*) – If True, also return the path to the TRNSYS dck file (.dck).
- **output_folder** (*str, optional*) – location where output files will be
- **trnsidf_exe** (*str*) – Path to *trnsidf.exe*.
- **template** (*str*) – Path to d18 template file.
- **log_clear_names** (*bool*) – If True, DOES NOT log the equivalence between the old and new names in the console.
- **schedule_as_input** (*bool*) – If True, writes the schedules as INPUTS in the BUI file. Then, the user would have to link in TRNSYS studio the csv file with the schedules to those INPUTS. If False, the schedules are written as SCHEDULES in the BUI file. Be aware that this last option (False) can make crash TRNBuild because the schedules are too long are there is too many schedules.
- **kwargs** – keyword arguments sent to *convert_idf_to_trnbuild()* or *trnbuild_idf()* or *choose_window()*. "ordered=True" to have the name of idf objects in the outputfile in ascendant order. See *trnbuild_idf()* or *choose_window()* for other parameter definition

Returns

A tuple containing:

- return_b18 (str): the path to the TRNBuild file (.b18). Only provided if *return_b18* is True.
- return_trn (str): the path to the TRNBuild input file (.idf). Only provided if *return_t3d* is True.
- retrun_dck (str): the path to the TRNSYS dck file (.dck). Only provided if *return_dck* is True.

Return type (tuple)

trnslator.trnsys.get_idf_objects

trnslator.trnsys.**get_idf_objects**(*idf*)

Gets idf objects

Parameters **idf** (*trnslator.idfclass.IDF object at 0x11e3d3208*) – the IDf object

Returns

MATERIAL object from the IDF materialNoMass (Idf_MSequence): MATERIAL:NOMASS object from the IDF materialAirGap (Idf_MSequence): MATERIAL:AIRGAP object from the IDF versions (Idf_MSequence): VERSION object from the IDF buildings (Idf_MSequence): BUILDING object from the IDF locations (Idf_MSequence): SITE:LOCATION object from the IDF globGeomRules (Idf_MSequence): GLOBALGEOMETRYRULES object from the

IDF constructions (Idf_MSequence): CONSTRUCTION object from the IDF buildingSurfs (Idf_MSequence): BUILDINGSURFACE:DETAILED object

from the IDF

fenestrationSurfs (Idf_MSequence): FENESTRATIONSURFACE:DETAILED object
from the IDF

zones (Idf_MSequence): ZONE object from the IDF peoples (Idf_MSequence): PEOPLE object from the IDF lights (Idf_MSequence): LIGHTs object from the IDF equipments (Idf_MSequence): EQUIPMENT object from the IDF

Return type materials (Idf_MSequence)

trnslator.trnsys.clear_name_idf_objects

trnslator.trnsys.**clear_name_idf_objects**(*idfFile*, *log_clear_names=False*)
Clean names of IDF objects.

Replaces variable names with a unique name, easy to refer to the original object. For example : if object is the n-th "Schedule Type Limit", then the new name will be "stl_00000n" - limits length to 10 characters

Parameters

- **idfFile** (trnslator.idfclass.IDF) – IDF object where to clean names
- **log_clear_names** –

trnslator.trnsys.zone_origin

trnslator.trnsys.**zone_origin**(*zone_object*)
Return coordinates of a zone

Parameters **zone_object** (*EpBunch*) – zone element in zone list.

Returns Coordinates [X, Y, Z] of the zone in a list.

trnslator.trnsys.closest_coords

trnslator.trnsys.**closest_coords**(*surfList*, *to=[0, 0, 0]*)
Find closest coordinates to given ones

Parameters

- **surfList** (*idf_MSequence*) – list of surfaces with coordinates of each one.
- **to** (*list*) – list of coordinates we want to calculate the distance from.

Returns the closest point (its coordinates x, y, z) to the point chosen (input "to")

trnslator.trnsys.parse_window_lib

trnslator.trnsys.**parse_window_lib**(*window_file_path*)

Function that parse window library from Berkeley Lab in two parts. First part is a dataframe with the window characteristics. Second part is a dictionary with the description/properties of each window.

Parameters **window_file_path** (*str*) – Path to the window library

Returns

a tuple of:

- dataframe: df_windows, a dataframe with the window characteristics in the columns and the window id as rows
- dict: bunches, a dict with the window id as key and description/properties of each window as value

Return type tuple

trnslator.trnsys.choose_window

trnslator.trnsys.**choose_window**(*u_value*, *shgc*, *t_vis*, *tolerance*, *window_lib_path*)

Return window object from TRNBuild library

Returns (tuple): A tuple of:

- window_ID
- window's description (label)
- window's design (width of layers)
- window u-value
- window shgc
- window solar transmittance
- window solar refraction
- window visible transmittance
- number of layers of the window
- window width
- the "bunch" of description/properties from Berkeley lab

If tolerance not respected return new tolerance used to find a window.

Parameters

- **u_value** (*float*) – U_value of the glazing given by the user
- **shgc** (*float*) – SHGC of the glazing given by the user
- **t_vis** (*float*) – Visible transmittance of the glazing given by the user
- **tolerance** (*float*) – Maximum tolerance on u_value, shgc and tvis wanted by the user
- **window_lib_path** (*dat file*) – window library from Berkeley lab

trnslator.trnsys.trnbuild_idf

trnslator.trnsys.**trnbuild_idf**(*idf_file*, *output_folder=None*, *template=None*, *dck=False*, *nonum=False*, *N=False*, *geo_floor=0.6*, *refarea=False*, *volume=False*, *capacitance=False*, *trnsidf_exe=None*)

This program sorts and renumbers the IDF file and writes a B18 file based on the geometric information of the IDF file and the template D18 file. In addition, an template DCK file can be generated.

Important: Where settings.trnsys_default_folder must be defined inside the configuration file of the package

Example

```
>>> # Exemple of setting kwargs to be unwrapped in the function
>>> kwargs_dict = {'dck': True, 'geo_floor': 0.57}
>>> # Exemple how to call the function
>>> trnbuild_idf(idf_file,template=os.path.join(
>>>              settings.trnsys_default_folder,
>>>              r"Building\trnsIDF\NewFileTemplate.d18"
```

Parameters

- **idf_file** (*str*) – path/filename.idf to the T3D file "a SketchUp idf file"
- **output_folder** (*str, optional*) – location where output files will be
- **template** (*str*) – path/NewFileTemplate.d18
- **dck** (*bool*) – If True, create a template DCK
- **nonum** (*bool, optional*) – If True, no renumeration of surfaces
- **N** (*optional*) – BatchJob Modus
- **geo_floor** (*float, optional*) – generates GEOSURF values for distributing direct solar radiation where *geo_floor* % is directed to the floor, the rest to walls/windows. Default = 0.6
- **refarea** (*bool, optional*) – If True, floor reference area of airnodes is updated
- **volume** (*bool, True*) – If True, volume of airnodes is updated
- **capacitance** (*bool, True*) – If True, capacitance of airnodes is updated
- **trnsidf_exe** (*str*) – Path of the trnsidf.exe executable

Returns status

Return type str

Raises **CalledProcessError** – When could not run command with trnsidf.exe (to create BUI file from IDF (T3D) file

1.10.7 Utils

config	Package configurations.
validate_trnsys_folder	**param trnsys_default_folder**
log	Write a message to the log file and/or print to the the console.
newrange	Takes the previous DataFrame and calculates a new Index range.
date_transform	Simple function transforming one-based hours (1->24) into zero-based hours (0->23)
weighted_mean	Compute the weighted average while ignoring NaNs.
top	Compute the highest ranked value weighted by some other variable.
copy_file	Handles a copy of test idf files
piecewise	returns a piecewise function from an array of the form [hour1, hour2, . . . , value1, value2, . . .]
rmse	calculate rmse with target values
checkStr	Find the first occurrence of a string and return its line number
write_lines	Delete file if exists, then write lines in it
check_unique_name	Making sure new_name does not already exist
angle	Calculate the angle between 2 vectors
float_round	Makes sure a variable is a float and round it at "n" decimals
timeit	Use this method as a decorator on a function to calculate the time it take to complete.
lcm	This function takes two integers and returns the L.C.M.
recursive_len	Calculate the number of elements in nested list
rotate	Shift list elements to the left
parallel_process	A parallel version of the map function with a progress btr.

trnslator.utils.config

trnslator.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='trnslator'*, *log_filename='trnslator'*, *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')*, *trnsys_default_folder=Path('C:\\TRNSYS18')*, *default_weight_factor='area'*, *ep_version='9-2-0'*)

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 many of trnslator outputs such as EnergyPlus simulation results. This can save a lot of time by not calling the simulation and DataPortal APIs repetitively 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.
- **trnsys_default_folder** (*str*) – root folder of TRNSYS install.
- **default_weight_factor** –
- **ep_version** (*str*) – EnergyPlus version to use. eg. "9-2-0".

Returns None

trnslator.utils.validate_trnsys_folder

trnslator.utils.**validate_trnsys_folder**(*trnsys_default_folder*)

Parameters **trnsys_default_folder** –

trnslator.utils.log

trnslator.utils.**log**(*message*, *level=None*, *name=None*, *filename=None*, *avoid_console=False*, *log_dir=None*)

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

trnslator.utils.newrange

trnslator.utils.**newrange**(*previous*, *following*)
Takes the previous DataFrame and calculates a new Index range. Returns a DataFrame with a new index

Parameters

- **previous** (*pandas.DataFrame*) – previous DataFrame
- **following** (*pandas.DataFrame*) – following DataFrame

Returns DataFrame with an incremented new index

Return type pandas.DataFrame

trnslator.utils.date_transform

trnslator.utils.**date_transform**(*date_str*)
Simple function transforming one-based hours (1->24) into zero-based hours (0->23)

Parameters **date_str** (*str*) – a date string of the form 'HH:MM'

Returns datetime object

Return type datetime.datetime

trnslator.utils.weighted_mean

trnslator.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

trnslator.utils.top

trnslator.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

trnslator.utils.copy_file

trnslator.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)

trnslator.utils.piecewise

trnslator.utils.**piecewise**(*data*)
returns a piecewise function from an array of the form [hour1, hour2, ..., value1, value2, ...]

Todo : write de description of the args :param data:

trnslator.utils.rmse

trnslator.utils.**rmse**(*data*, *targets*)
calculate rmse with target values

Todo : write de description of the args :param data: :param targets:

trnslator.utils.checkStr

trnslator.utils.**checkStr**(*datafile*, *string*, *begin_line=0*)
Find the first occurrence of a string and return its line number

Returns: the list index containing the string

Parameters

- **datafile** (*list-like*) – a list-like object
- **string** (*str*) – the string to find in the txt file

trnslator.utils.write_lines

trnslator.utils.**write_lines**(*file_path*, *lines*)
Delete file if exists, then write lines in it

Parameters

- **file_path** (*str*) – path of the file
- **lines** (*list of str*) – lines to be written in file

trnslator.utils.check_unique_name

trnslator.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)

trnslator.utils.angle

trnslator.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)

trnslator.utils.float_round

trnslator.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)

trnslator.utils.timeit

trnslator.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.

trnslator.utils.lcm

trnslator.utils.**lcm**(*x*, *y*)
This function takes two integers and returns the L.C.M.

Parameters

- **x** –
- **y** –

trnslator.utils.recursive_len

trnslator.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

trnslator.utils.rotate

trnslator.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

trnslator.utils.parallel_process

trnslator.utils.**parallel_process**(*in_dict*, *function*, *processors=- 1*, *use_kwargs=True*)
A parallel version of the map function with a progress btr.

Examples

```
>>> import trnslator as tr
>>> 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'
>>> files = tr.copy_file(files)
>>> rundict = {file: dict(eplus_file=file, weather_file=wf,
>>>                       ep_version=ep_version, annual=True,
>>>                       prep_outputs=True, expandobjects=True,
>>>                       verbose='q', output_report='sql')
>>>           for file in files}
>>> result = parallel_process(rundict, tr.run_eplus, use_kwargs=True)
```

Parameters

- **in_dict** (*dict-like*) – A dictionary to iterate over.
- **function** (*function*) – 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* .

Returns [function(array[0]), function(array[1]), . . .]

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

Z