NIST SURF 2021 Research Summary

National Institute of Standards and Technology, Summer Undergraduate Research Fellowship, Engineering Laboratory, Smart Grid and Cyber-Physical Systems

[Brian Woo-Shem](http://www.brianwooshem.com/)

Santa Clara University | Mechanical Engineering B.S. ‘21, M.S. ‘23

Website: [www.brianwooshem.com](http://www.brianwooshem.com)

August 12, 2021

1. Occupancy Thermostat Simulations

Research project of performing occupancy and adaptive control simulations to demonstrate the energy saving capabilities of these algorithms over traditional fixed setpoint control. Thermal comfort of occupants is considered as well. Optimization using price and predicted temperature was attempted but did not yield energy savings over adaptive setpoint or occupancy setpoint methods. Simulations were run for winter and summer models using varying electric prices based on real wholesale prices in the region.

2. SURF Presentation

A final presentation about my research work on improving the efficiency of heating and air conditioning systems with optimized thermostat algorithms. In a series of simulations, I show that thermostats that adapt to the outdoor temperature and to occupancy of a building can save an expected 36% over conventional thermostats.

**Slides:** <https://github.com/bwooshem/Engineering-Portfolio/blob/main/SURF_Presentation_BrianWoo-Shem_Publish.pdf>

3. Occupancy-Based Control Simulation Code

Implemented real-time occupancy and probability of occupancy into the HVAC optimization co-simulation. This builds on the adaptive control, optimizer, UCEF, & EnergyPlus simulation project. Simulation consists of two federates, a socket to communicate with EnergyPlus, and a controller using a Python optimization and computation code. The main simulation is run in the UCEF VM using procedures in the [wiki](https://github.com/SCU-Smart-Grid-CPS/smart-grid-energy-simulation-research/wiki), and the EnergyPlus component is run in Windows using Joe\_ep\_fmu.

**Files**: <https://github.com/SCU-Smart-Grid-CPS/TwoFedEPOpt>

**Full Documentation:** See page 4

4. CAISO Pricing Data

Python script with command-line inputs to download any number of days that exist of Real-Time and Day-Ahead electric market pricing from the California Independent System Operator. Converts data of different timesteps to 5-minute timesteps.

Output files can be put directly into the co-simulation and read by the optimization and occupancy-setpoint codes.

**Code**: <https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/getWholesaleCAISO.py>

**Example Result**:

<https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/WholesaleRealTime_2019-12-21_2020-3-30.csv>

**Full Documentation:** See page 11

5. Converting Real Historical Weather Data to EnergyPlus Weather Files

Creates a full-year .epw file for EnergyPlus simulation using real-life recorded weather for any recent year. Identified open datasource called [MesoWest](https://mesowest.utah.edu/), developed a combination procedure to quickly download year-long datasets, use a Python code to process the data from semi-random intervals into hourly timesteps, and then modify .epw files using [Elements](https://bigladdersoftware.com/projects/elements/) software.

Procedure

**Code**: <https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/mesoweatherepw.py>

**Full Documentation:** See page 12

6. Simulation Output Post-Processing

Python code and Excel spreadsheet for quickly analyzing and plotting the results of simulations. Can compute the percentage of occupied time that the space was occupied and comfortable, total electricity used for heating and cooling, and metrics for how comfortable the building is overall. Produces visualizations of the setpoints, indoor temperature, energy usage data, etc.

**Code**: <https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/epPostProcess.py>

**Example output:** <https://github.com/SCU-Smart-Grid-CPS/smart-grid-energy-simulation-research/blob/Tutorials/screenshots/Optimized%20Occupancy%20Control%20Behavior%20-%20Summer%2C%20Sacramento%20June%2029%20-%20July%205_labeled.png>

7. UCEF, EnergyPlus & Python Documentation

Built GitHub wiki containing guides for using UCEF, WebGME, removing time delays in co-simulations, using a functional mock-up unit interface to communicate between EnergyPlus and UCEF, EnergyPlus reference guides, the procedure for inputting historical weather to EnergyPlus, simulation-specific Python issues, and more. It is intended for training future students in this program, but is publicly available to help other researchers.

**Link**: <https://github.com/SCU-Smart-Grid-CPS/smart-grid-energy-simulation-research/wiki>

8. Unified GitHub Organization & Repository

Organized a disparate collection of GitHub repositories for the project code held by individual students into a unified organization. This will ensure that the code is accessible to future researchers and not lost when students leave the program.

**Link**: <https://github.com/SCU-Smart-Grid-CPS>

9. Belonging Initiative Presentation: Investigating Inequality in Health

Presentation to the Belonging Initiative at NIST. Overview of historical effects of racism and slavery on modern health outcomes. Demonstration through demographic statistics how health, education, and environmental issues are contributors that perpetuate inequality. A discussion of potential solutions using science and engineering.

**Slides Archive**: <https://github.com/bwooshem/Engineering-Portfolio/blob/main/Investigating_Inequality_in_Health_Brian_Woo-Shem_July_2021.pdf>

Energy Plus Occupancy + Optimization 2-Federate Simulation

Version 5.11 BETA - new feature development release

Brian Woo-Shem

Updated: 2021-08-11

This is a minor update; only the deployment folder needs to be updated from version 5.01

**Support for** [**getWholesaleCAISO**](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing) **and** [**getWeatherSolar**](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/tree/main/GetWeatherSolar)**:**

This version is primarily for those using [getWholesaleCAISO](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing) and [getWeatherSolar](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/tree/main/GetWeatherSolar) for data inputs instead of Solar.xlsx, WholesalePrice.xlsx, and OutdoorTemp.xlsx.

If using exclusively the older method, it is safer to keep using version 5.10.

Both methods are supported, using the [legacy](https://github.com/SCU-Smart-Grid-CPS/TwoFedEPOpt/blob/5bbde3a8429706cb9cbe183efbfba6a0471ae3fa/EnergyPlusOpt2Fed/EnergyPlusOpt2Fed_deployment/energyOptTset2hr.py#L71) variable in energyOptTset2hr.py and occupancyAdaptSetpoints.py. Set legacy = True if using the old method.

**Faster Runtime:**

In the Python optimization and adaptive/occupancy setpoints codes, using numpy for input weather, wholesale price, and solar radiation instead of pandas dataframes reduces total simulation runtime significantly on the VirtualBox VMs - from 52 minutes with dataframes to 12 minutes with numpy on one test computer. The differences are smaller but still noticeable on the cluster.

Version 5.01 - Stable feature and bug fixing release

Updated: 2021-07-26

This is a major update; downloading and using the new federation files is recommended, rather than selectively swapping on an older version.

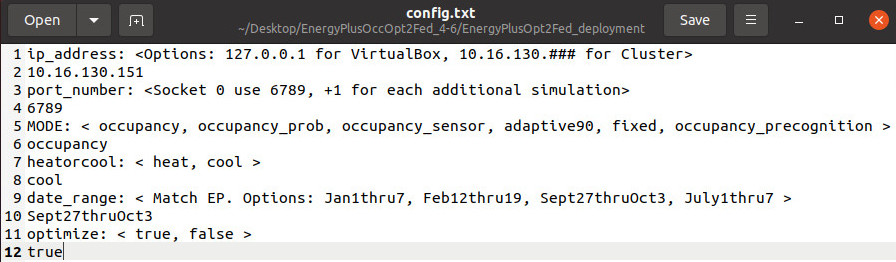
**New Optimization & Occupancy Modes:**

The federation now supports 12 different operation modes:

1. Occupancy probability & current status + Optimization: the primary operation mode. Optimization combining probability data and current occupancy status
2. Occupancy probability & current status (no optimization)
3. Occupancy probability + optimization: optimization with only occupancy probability (NOT current status)
4. Occupancy probability (no optimization)
5. Occupancy current status + optimization: optimization with only occupancy sensor data for current occupancy status
6. Occupancy current status (no optimization)
7. Adaptive Setpoints + Optimization: optimization with adaptive setpoints where 90% people are comfortable. No occupancy
8. Adaptive Setpoints (no optimization)
9. Fixed + Optimization: optimization with fixed setpoints. No occupancy.
10. Fixed (no optimization)
11. Occupancy Prescheduled + Occupancy: Optimize if occupancy status for the entire prediction period (2 hrs into future) is known.
12. Occupancy Prescheduled (no optimization)

**Easy Configuration:**

* config.txt is now located in the deployment folder.
* Previously, to change the mode, heating vs. cooling, date range, and optimization settings, the Controller.java and energyOptTset2hr.py files had to be edited and recompiled with bash build-all.sh. Now, the code files stay the same and parameters are set in a single config.txt
* See instructions below



Note: do NOT include quote marks ‘’ around strings

**MODE**:

# 'occupancy': the primary operation mode. Optimization combining probability data and current occupancy status

# 'occupancy\_prob': optimization with only occupancy probability (NOT current status)

# 'occupancy\_sensor': optimization with only occupancy sensor data for current occupancy status

# 'adaptive90': optimization with adaptive setpoints where 90% people are comfortable. No occupancy

# 'fixed': optimization with fixed setpoints. No occupancy.

# 'occupancy\_precognition': Optimize if occupancy status for entire prediction period (2 hrs into future) is known. A joke!?

**heatorcool**:

# 'heat': only heater, use in winter

# 'cool': only AC, use in summer

**date\_range:**

# 'Jan1thru7'

# 'Feb12thru19'

# 'Sep27-Oct3\_SJ'

# 'July1thru7'

# 'bugoff': For debugging and testing specific inputs. Hot w rapid cool off. Run with day = 1, hour = 0.

# 'bugfreeze': Extreme cold values, rapidly gets colder, for testing. Run with day = 1, hour = 0.

# 'bugcook': Extreme hot values, cools briefly then gets hotter for testing. Run with day = 1, hour = 0.

# 'bugsnug': Comfortable 18-23°C for testing both heat and cool.

# 'bugwarm': Less extreme hot than bugcook mode

# 'bugAC': Figure out why cool mode keeps failing if the price changes

# 'bughprice': Analogous to bugAC but for heating with price change

**optimize**:

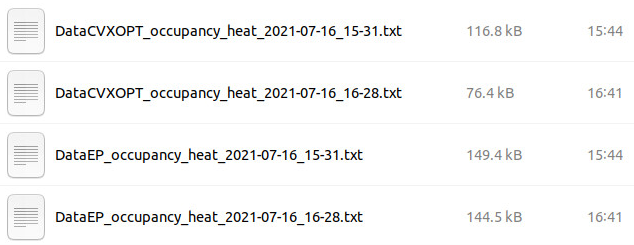
true: Use optimization code, energyOptTset2hr.py and try to optimize usage

false: Use the previously specified mode, but do not optimize

**Well-Behaved File Paths**

File paths in energyOptTset2hr.py, Controller.java, and Socket.java are now routed to the deployment folder. There is no need to change filepaths in the code when moving to a new computer.

Data output files are labeled with settings, date, and time so that they don’t overwrite each other and it is easier to determine what each one is for afterward.



**energyOptTset2hr.py Improvements**

**Occupancy Integration**: Can now optimize for four occupancy modes (see 1, 3, 5, 11 under “New Optimization & Occupancy modes)

* Gets occupancy data from occupancy\_1hr.csv
* Optimizes using setpoint bounds determined by occupancy
* WARNING: old versions of the occupancy\_1hr.csv may not work correctly

**New Input Parameters Method:** date range, heat or cool, mode, number of timesteps to optimize, and number of timesteps to return can all be set either as values at the start of the program, or through additional inputs when running the code. This is meant for future controller improvements such that optimization can be run more frequently, and heating vs. cooling is set automatically by the program during operation. Including these parameters are optional and they can still be set within the python script as before.

The following formats all will work:

python energyOptTset2hr.py day hour temp\_indoor\_initial

python energyOptTset2hr.py day hour temp\_indoor\_initial n nt

python energyOptTset2hr.py day hour temp\_indoor\_initial n nt heatorcool

python energyOptTset2hr.py day hour temp\_indoor\_initial n nt heatorcool MODE

python energyOptTset2hr.py day hour temp\_indoor\_initial n nt heatorcool MODE date\_range

python energyOptTset2hr.py day hour temp\_indoor\_initial heatorcool

python energyOptTset2hr.py day hour temp\_indoor\_initial heatorcool MODE

python energyOptTset2hr.py day hour temp\_indoor\_initial heatorcool MODE date\_range

python energyOptTset2hr.py day hour temp\_indoor\_initial heatorcool nt

python energyOptTset2hr.py day hour temp\_indoor\_initial n MODE

python energyOptTset2hr.py day hour temp\_indoor\_initial n MODE date\_range

Note: Linux may use 'python3' or 'python3.9' instead of 'python'

Windows use 'py'

day = [int] day number in simulation. 1 =< day =< [Number of days in simulation]

hour = [int] hour of the day. 12am = 0, 12pm = 11, 11pm = 23

temp\_indoor\_initial = [float] the initial indoor temperature in °C

n = [int] number of 5 minute timesteps to use in optimization. Requirement: n >= nt

nt = [int] number of 5 minute timesteps to return prediction for. Requires: n >= nt

heatorcool = [str] see ===> SET HEATING VS COOLING! <===

MODE = [str] see ===> MODE <===

date\_range = [str] see ===> WHEN TO RUN <===

**Other improvements**

* Added switching for adaptive vs fixed vs occupancy control - Brian
* Added occupancy optimization - Brian
* Fixed optimizer should not know occupancy status beyond current time
* Merging 2 codes, keeping the best characteristics of each
* Base code is energyOptTset2hr\_kaleb.py, with stuff from the PJ/Brian one added
* Runtime reduced from 2.0 +/- 0.5s to 1.0 +/- 0.5s by moving mode-specific steps inside "if" statements and taking dataframe subsets before transformation to matrices
* Both heat and cool modes can optimize without out of bounds errors.
* Fixed off by one error where indoor temperature prediction is one step behind energy, causing it to miss some energy savings

**occupancyAdaptSetpoints.py**

New code that returns the non-optimized setpoints for any of the occupancy, adaptive, or fixed modes. It is a subset fork of the optimizer code, and it functions the same way but does not do any of the optimization stuff. Theoretically it can replace the section in Controller.java that computes adaptive and fixed un-optimized setpoints, but the simulation would run slower in these cases because it has to send to/from the Python.

Run it the same way as energyOptTset2hr.py

**Controller.java**

* Inputs for the heat or cool, mode, date range, and optimization now come from config.txt, see above.
* Added compatibility with occupancyAdaptSetpoints.py
* Added use config file instead of changing variables in here to reduce recompiling
* Fixed datastring receiver from Python so that datastrings do not continually append with data for the next datastring, and entire process is much faster and more reliable
* Cleaned up code
* Has variables needed for more frequent optimization calls but does not work because of energyOptTset2hr limitations
* No more long filepaths to change! Instead it saves to the run directory (usually deployment) by default!
* EP and CVXOPT Data Summary file paths have a date + time format so they don't overwrite previous runs.
* Includes more coherent error messages for common issues:
  + Python crash
  + Lost socket connection
  + Failed config input
  + Failed write to file

**Socket.java**

The filepath for the config.txt goes to deployment. No need to change and recompile on new computers.

**Auto Shut-down in UCEF Federates**

Socket and controller previously would print “Waiting for \_\_ interaction” forever until they crash if the other federate closes. Now, they do 200 attempts over about 80 seconds to wait for the other’s interaction; if they fail to respond after this long, it usually means the other is closed or crashed. After the wait period, the remaining federate also shuts down.

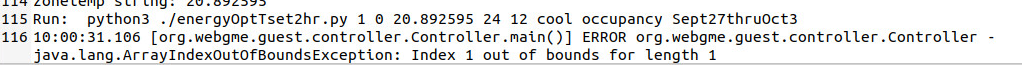
**Tips & Tricks**

Date ranges in EP are inclusive.

|  | Jan1thru7 | Feb12thru19 | Sept27thruOct3 |
| --- | --- | --- | --- |
| Start Month | Jan | Feb | Sept |
| Start Day | 1 | 12 | 27 |
| End Month | Jan | Feb | Oct |
| End Day | 1 | 18 | 3 |

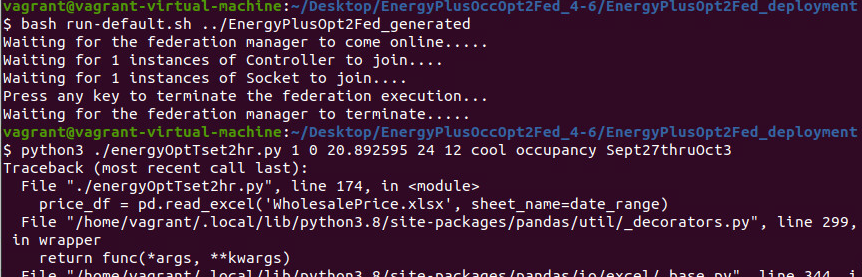
If end day Feb 19 is used in Feb12thru19, EP will try to use an eighth day and the weather, WholesalePrice, Solar files in UCEF will not have enough data so either Controller or the Python will crash over a data not found error.

If it crashes on an optimization or occupancy mode, check the last lines of the controller log. If you see something like this:

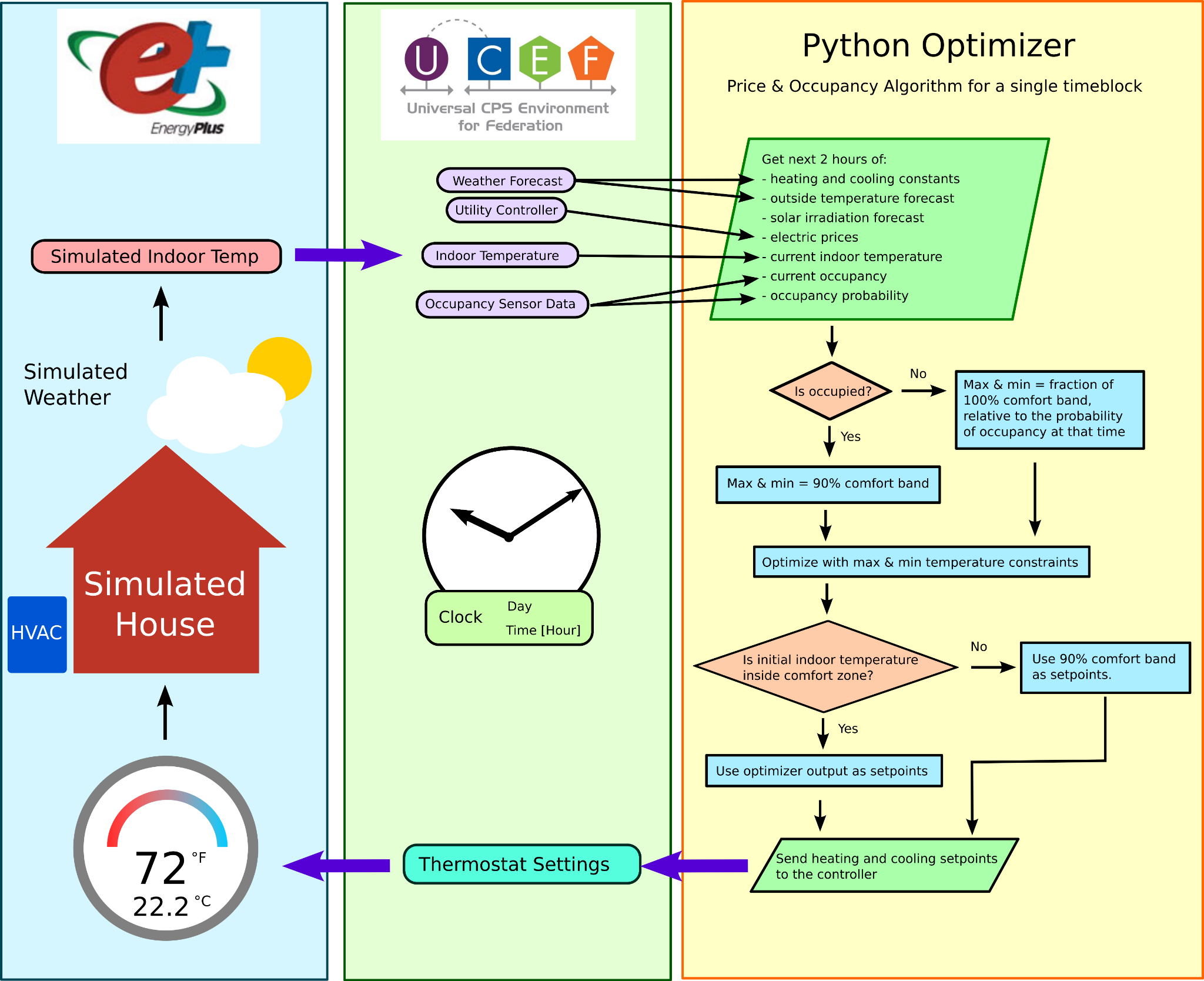


It indicates that Python likely crashed and did not return values, so when Java tries to use the array of values sent, it returns null and Java crashes. The problem is in Python.

To debug, take the line after “Run:” and paste it into the same terminal where you ran the simulation. Use the Python output to debug the Python code. When the command runs without crashing, then the simulation will probably work.



**Simulation Components and Process**



**UCEF 2-Federate Version**

The original simulation was simplified to 2 key federates to reduce complexity.

**Socket**: Connects via TCP/IP to the EnergyPlus simulation, and transfers the indoor temperature setpoints and building conditions back and forth.

**Controller**: Using indoor temperature conditions, computes the ideal setpoints according to the configured method.

How to get CAISO Day Ahead Electricity Market Price Data

Version 1.0 - Stable, Full Functionality

Brian Woo-Shem

Updated: 2021-07-28

Automatically get any number of days CAISO wholesale price data in Python.

Simply run this program with type of data and what dates as shown below, and it will generate a .csv with wholesale price data in a single column at 5 minute timesteps that can be plugged directly into UCEF and the optimizer script.

Outputs csv with file name in format: WholesaleRealTime\_2019-12-21\_2020-3-30.csv

New script [getWholesaleCAISO.py](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/mesoweatherepw.py)

SYNTAX:

getWholesaleCAISO.py <data\_type> <year> <month> <day> <number\_of\_days> <optional\_options>

<data\_type> d for day ahead, r for real time

<year> four digit year of first day to get (eg. 2020)

<month> month of the first day to get (eg. January, Jan, or 1)

<day> day of month for the first day (eg. 21)

<number\_of\_days> how many days to get data for. 1 = only get the day specified

<optional\_options> Additional parameters, optional

-v verbose: provide extra outputs for user readability

-d debug: provide detailed outputs for debugging

-leap include leap years in date calculation

-s Server Safe mode; adds delay to avoid crashing server but takes longer.

-fast fast unsafe: removes all time delays for fastest data acquisition. WARNING: you are responsible if this mode breaks something or violates the CAISO Terms of Use.

For licensing info, run

getWholesaleCAISO.py -l

Required Modules: beautifulsoup4, urllib3, numpy, os\_sys, lxml, requests

**Example Input:**

Returns week starting Aug 3, 2020 with day-ahead values

getWholesaleCAISO.py d 2020 8 3 7

Create Full Year EnergyPlus Weather Files with Real Historical Weather Data

Version 0.6 BETA

Brian Woo-Shem

Last updated: 2021-08-11

Get historical weather data for any recent year for many locations around the U.S. and world, process it into hourly averages when data is more frequently reported, and paste into the Energy Plus weather file for simulations.

This process works but is only 50% automated. Weather for 1 year at 1 location takes 30 minutes to 1 hour to set up.

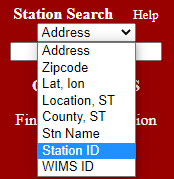
# 1. Get Source Weather Data

Start with an epw weather file from:

<http://climate.onebuilding.org/WMO_Region_4_North_and_Central_America/USA_United_States_of_America/index.html#IDCA_California->

Go to MesoWest: <https://mesowest.utah.edu/>

On MesoWest, change search type to station id



Search the airport name, matching the one the EPW is from.

Note: If needed, find the airport codename (3 letters) from here: <https://www.airportcodes.us/us-airports-by-name.htm>

Then add a “K” in front. For example, San Jose International Airport is codenamed “SJC”, so we would search using “KSJC”



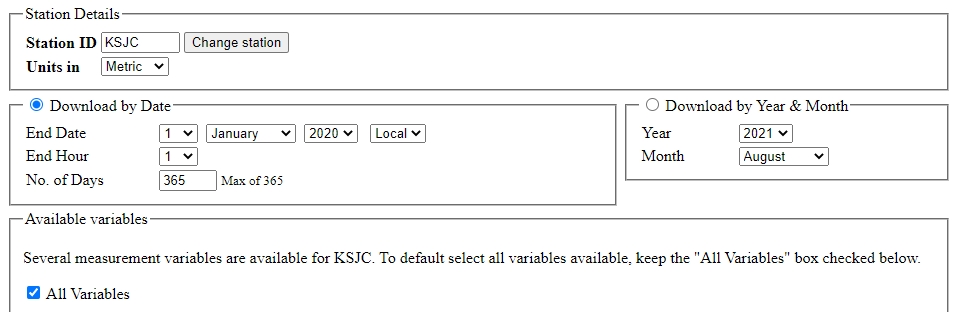
On left panel, look for Download Data



To download an entire year of data, you need to make a free account

**Use the following settings:**

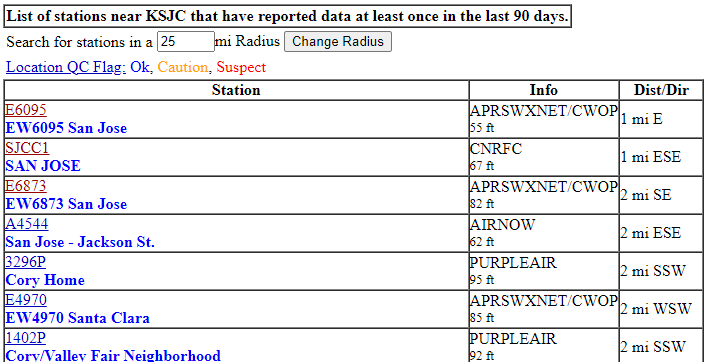
1. DO NOT change Station ID
2. Switch Units to Metric
3. The interface is clunky and asks for END date. Data is on hours 0-23.
4. To download an entire year at once
   1. End Date = the first day of January for the year AFTER the one you want. (ex: to get the entirety of 2019, set End Date = 1 January 2020.)
   2. End hour = 1.
   3. Number of days = 365
5. Make sure Download by Date is selected (should be by default)
6. Keep “All Variables” selected



Scroll to the bottom and . Beware it can take several minutes.

However, the airports don’t have Solar Radiation. Use the bottom left menu to select Nearby Stations.

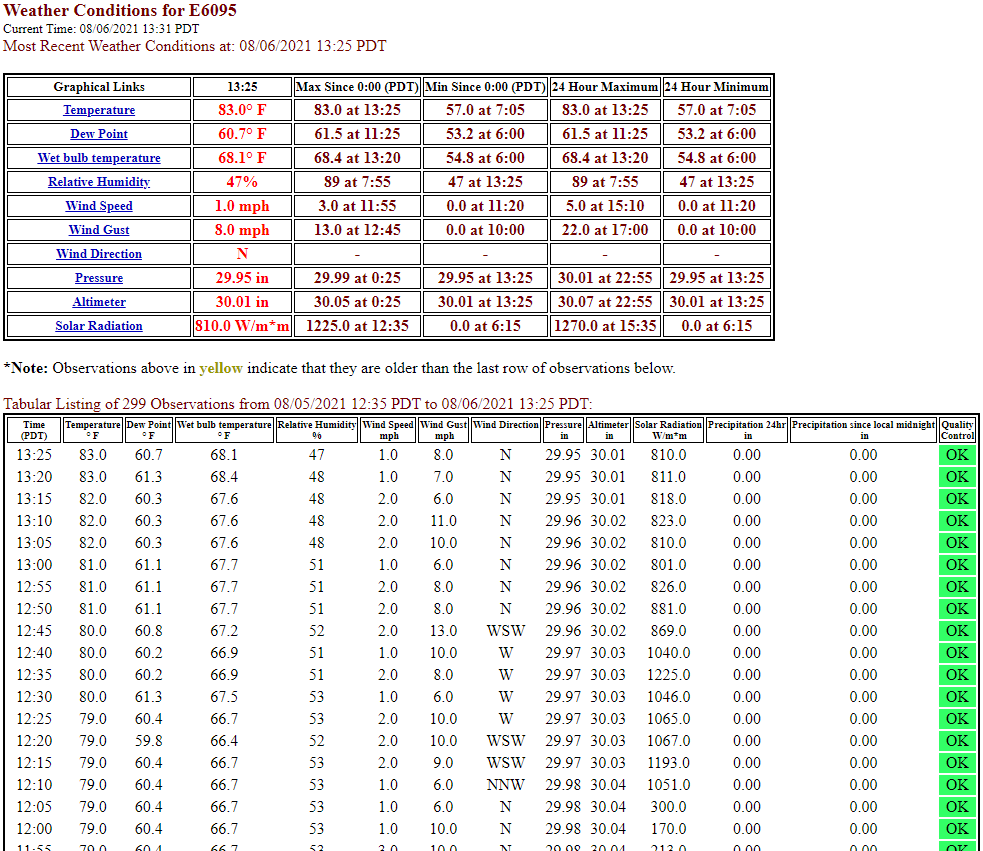
Look for a station that is of type **APRSWXNET/CWOP**



In this case, either the first or third entries will work. Pick the nearest one

On the page with tons of data, look to see how often the station is reporting, what variables it has, and if it is up to date. A good station reports at least every 15 minutes reliably, and has Solar Radiation values.

This is an example of a good station to pick: 5 minute timesteps, and within 1 mile



If the nearest one lacks good data, try the next nearest, and repeat until you find one that has decent data.

Repeat the download process for the same year of data from this station.

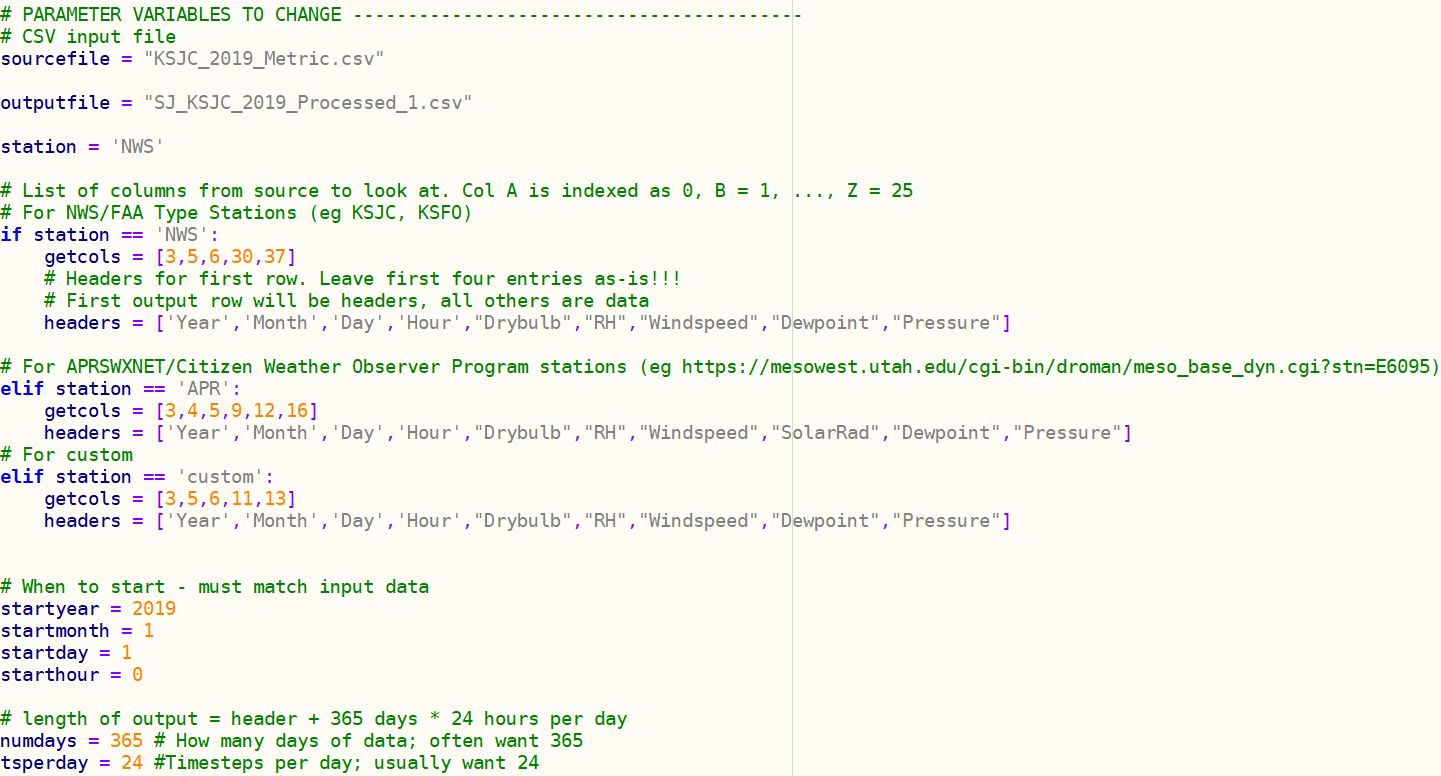
**Note**: Leap years are poorly supported. EPW does not have leap day, but downloading 365 days of a leap year from Mesowest will include leap day and skip Jan 1. To do a leap year, download an extra 1-2 days around Jan 1 separately, manually append it to the main input csv with the rest of the year, and manually delete Feb 29.

# 2. Process the CSV data from MesoWest

Use [mesoweatherepw.py](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/blob/main/mesoweatherepw.py)

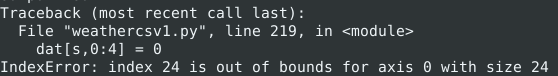
Change the variables in the top section before running the program.

1. Sourcefile must be the name of the input csv
2. Outputfile is the file it will write to
3. Station tells it what type of station. To process the airports (ie KSJC), use ‘NWS’. For the stations for solar radiation, use ‘APR’
4. Start year, month, day, and hour must match the first data row on the input csv
5. Change numdays if you want anything other than 1 year of data



Run the program in Python. It may take several minutes.

**Warning**: Data sets often have missing data or anomalies that will cause the program to crash. These must be resolved manually by editing the csv to something correct. Of 4 csvs, 3 had fatal errors initially.

* If input csv has a missing hour, the code can correct by filling in with the closest available data.
* If multiple consecutive hours are missing, the result will be inaccurate
* If input has duplicated time data, it will not crash and instead skip it. This error usually indicates that an hour is duplicated. Read the input above to figure out where in the file
* 
* If input is too short, it will give a warning. If it only gives the warning for the last hour of the year, this is expected and okay.

# 3. Merge into EPW

Download the free open-source Elements program from <https://bigladdersoftware.com/projects/elements/>

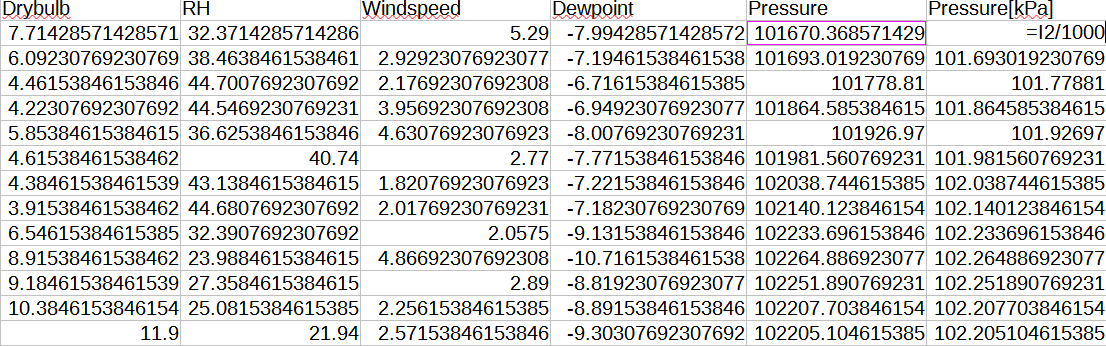
For help with Elements, see:

<https://bigladdersoftware.com/projects/elements/docs/user-guide/document-window.html>

Open the processed csv files for the airport and the APRSWXNET/CWOP stations.

**Airport**:

Next to Pressure create a new column. Call this Pressure[kPa] and set it to divide the pressure in Pa by 1000. Ctrl+D to do this for the entire length of the pressure values.



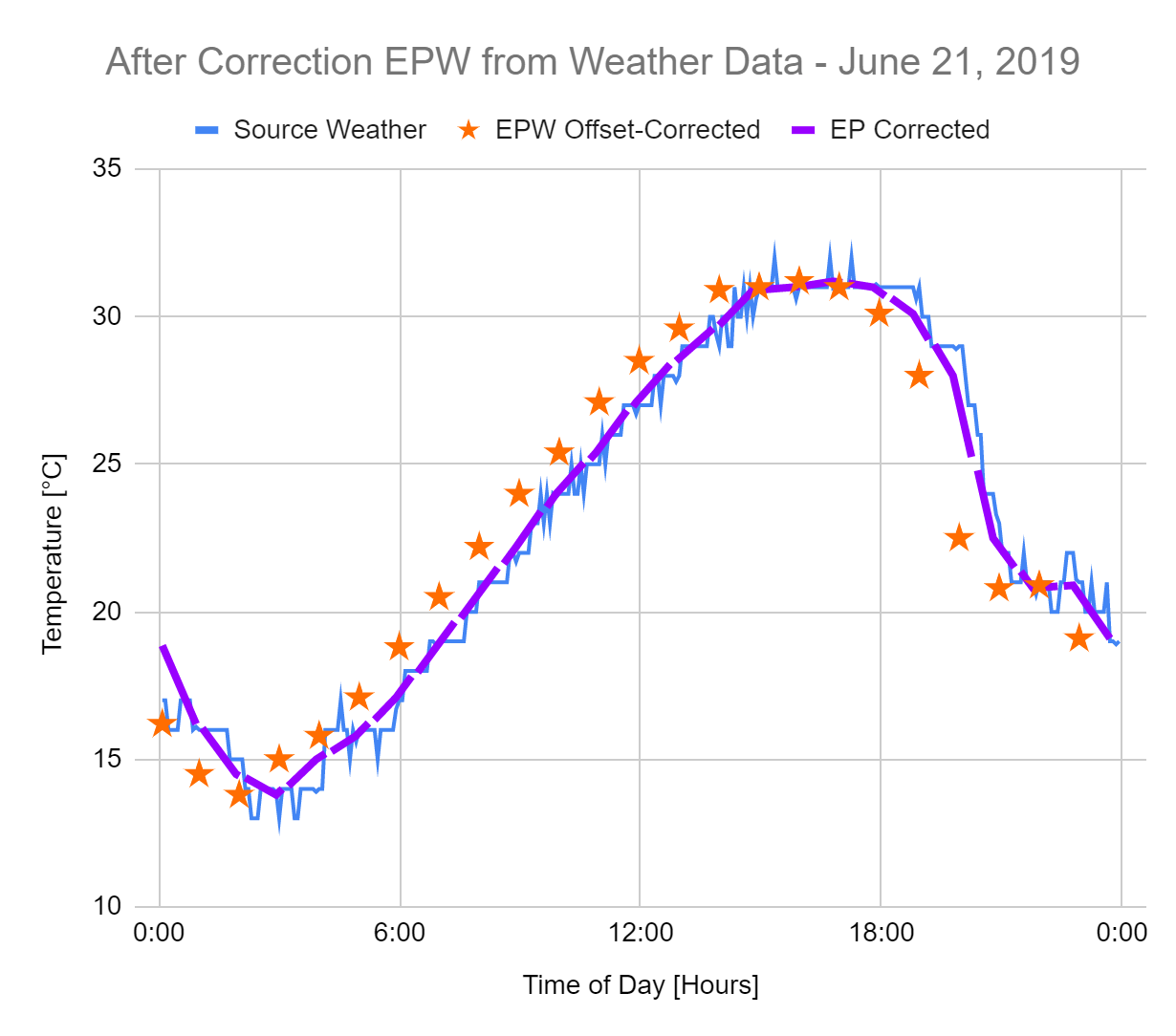
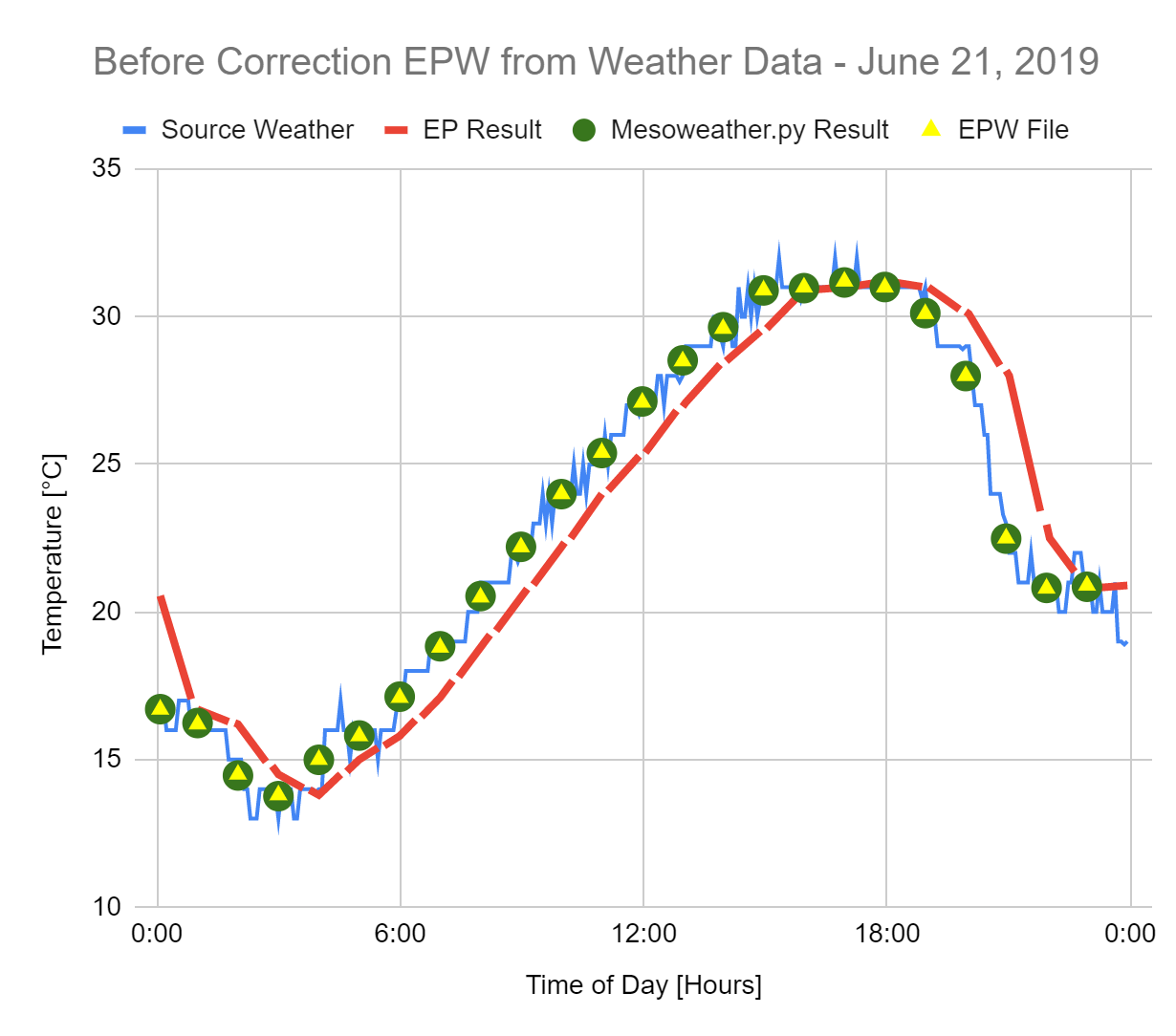
Open Elements

Copy paste the following columns from the processed csv to the Elements window by selecting the cells for one column that contain data, NOT the header and NOT any blank rows at the bottom.

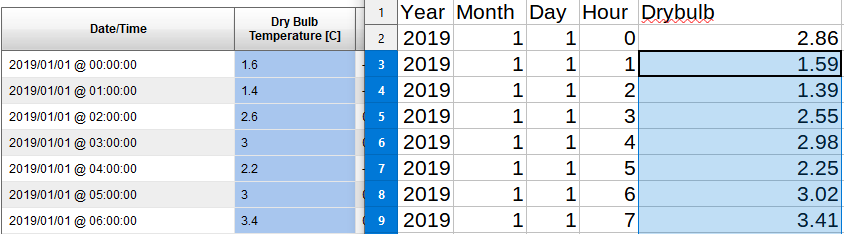
* Drybulb
* Relative Humidity (RH)
* Wind Speed
* Dew Point Temp
* Pressure [kPa]

**APRSWXNET/CWOP Station:** copy the Solar Radiation to Normal Solar in Elements.

**Update on weather and timesteps**: EnergyPlus has an offset issue such that if you take the Mesoweather output, copy it to the EPW matching the correct hours, and use this EPW, the EnergyPlus output will be one hour delayed (see first graph). However, the Mesoweather and EPW will match the original dataset.



To fix this, skip the 0th hour on the first day of data in Mesoweather output when copying, and instead paste hour 1 into 00:00:00 in Elements. This makes the EPW data appear to be offset relative to the source data, but gives the correct result in the simulation.



Using the “Header” button on top right and pop up window, change header to reflect new dataset. I often include the following:

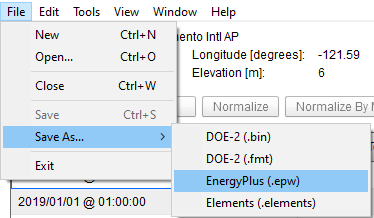
Comments (line 1) : year used, “Data from MesoWest open weather database” and list the station codenames and which data I got from which station

Comments (Line 2) add in front, replacing any download site credits: “Developed by Santa Clara University School of Engineering, Dept. of Mechanical Engineering.”

Reference Year = year used

Data Source = MesoWest/SCU

When finished, Save As an .epw file



Test by running a local simulation such as [GetWeatherSolar](https://github.com/SCU-Smart-Grid-CPS/Energy-Co-Simulation-Pre-and-Post-Processing/tree/main/GetWeatherSolar) before doing a UCEF one in case there are any bugs.