ASHRAE and IBPSA-USA SimBuild 2016
Building Performance Modeling Conference
Salt Lake City, UT
August 8-12, 2016

USING PYTHON AND EPPY FOR A LARGE NATIONAL SIMULATION STUDY

Jason Glazer, P.E.
GARD Analytics, Inc. Arlington Heights, IL.

ABSTRACT

A large study was performed using EnergyPlus to assess the maximum technically achievable energy targets for commercial buildings. The study looked at 16 different building models across 17 climates based on the PNNL prototype building models and applied 30 different "maxtech" energy efficiency measures to each. In order to perform the analysis, scripts were developed to modify each building model to implement each individual energy efficiency measures using the open source Eppy scripting library based on Python. In addition, Python was used to automatically extract results, analyze error messages, manage the selection of measure options, and produce summary tables. Simulations were run on a variety of computers, some local and some on the cloud, and managed by a Python script to optimize the use of resources. Data from all simulations was sent to a central location with remote monitoring. The strengths and weaknesses of the overall approach will be discussed as well as lessons learned by performing the overall analysis, the use of EnergyPlus, Eppy and Python scripting. The results of the analysis are presented in the final report and in a future paper. This paper focuses on the use of scripting to perform the analysis.

INTRODUCTION

This paper presents some background on the study performed and how it was performed, but the focus of the paper is to convey some pragmatic lessons learned during the project. Due to this, small summaries of various aspects of the project are presented along with discussion on how each aspect could be done better. Each summary discusses the strengths, weaknesses and lessons learned, when applicable. Many of the lessons learned make sense in the context of a large national study and may not be applicable to a specific building design project.

THE NATIONAL STUDY

The ASHRAE 1651-RP project titled "Development of Maximum Technically Achievable Energy Targets for Commercial Buildings - Ultra-Low Energy Use Building Set" was attempting to answer the question:

How energy efficient can commercial and multifamily buildings become in the near future if first cost is not considered?

The project used building energy simulation modeling to answer this question. The first step assembled a list of almost 400 energy efficiency measures that can be included in the design of commercial and multifamily residential buildings. Commonly used and cutting edge energy efficiency measures were included with the goal of being comprehensive, at least for measures that can be modeled. From this list, thirty were chosen to be modeled. Sixteen EnergyPlus prototype buildings that were consistent with ASHRAE Standard 90.1-2013 across all seventeen climate zones from the standard were used as baseline models (Halverson 2014). The measures were individually modeled using an open source Python based library called Eppy (Philip 2015), created for the study. Each of the thirty measures, often with many options, were applied to each building and climate combination. In general, the measures were applied in the following order:

- Reduce internal loads
- Reduce building envelope loads
- Reduce HVAC distribution system losses
- Decrease HVAC equipment energy consumption
- Major HVAC reconfigurations.

EnergyPlus was used to perform all of the building energy simulation modeling for this analysis. It had the capability of modeling nearly all of the almost 400 measure found during the first step of the project either directly or by using a thermally equivalent model or some other workaround. During the course of the project EnergyPlus was updated several times. Each individual measure was initially developed and tested using version 8.0 or 8.1, when the measures were combined, 8.3 (EnergyPlus 2015) was used.

Each measure was applied to each of the 272 building and climate combinations. If the energy consumption was reduced for a given building and climate, then the measure remained in the model. After all thirty measures were applied, the projected U.S. national weighted energy consumption for new buildings was nearly cut in half compared to ASHRAE Standard 90.1-2013. Since many of the measures had multiple options, over 14,000 simulations were run for the main scenario. To examine the incremental impacts of measures on the savings of later measures, five other scenarios were also simulated with various zone load measures removed. In total almost 69,000 simulations were run to achieve the final results. Many times that were performed during the trial and error portion of the project.

This paper does not focus on the results of the study, which can be found at Glazer (2016), but instead on the system used to perform the study including the use of Python and the Eppy library.

AUTOMATION

Automation, primarily using the Python scripting language, was used extensively to perform the analyses. The individual measures were implemented using a Python library called Eppy developed for the project that manipulates EnergyPlus input files. Management of the large number of simulations also used Python as did the extraction of results from the simulation output files and the formatting of those results into tables that could be used for generating reports. Simulation errors were collected and summarized automatically to aid the complex job of correcting problems.

Parametric analyses using building energy simulation generally use one of three different methods:

1. An input file is developed and individually edited for each permutation of building and climate for each energy efficiency measure.
2. Scripting embedded in a smaller set of input files enable and disable portions of the files depending on the measure and climate so that many permutations can be represented by each file.
3. A series of external scripts are developed where each script generically applies an energy efficiency measure revising the input file and the script is applicable across the various climates and buildings being modeled.

The first option would have been too time consuming for even a small parametric analyses and was not feasible for the scale of analysis in this project. The second option, embedded scripting has been used for many previous analyses, but it would not have been the best approach for this analysis. It is best for an analysis that has more buildings but fewer options for each building, because extra logic is needed for each combination of energy efficiency measures. In this analysis the number of possible combinations of energy efficiency measures was quite large, and the extra logic to support those combinations would have made such an approach difficult.

The third option was chosen and used scripts to read the existing prototype building model input files and produce revised input files with the energy efficiency measures added. The scripts were written in the programming language Python, using the Eppy library which was developed as part of this project. The scripts work with any of the prototype building model files and could be adapted to be used with any EnergyPlus input file. The revised input files have specific objects and fields modified, added, or removed to implement the energy efficiency measure. To include multiple measures in a file, the scripts were run sequentially. This scripting approach was well suited for the need to find the combination of measures that result in the building with the lowest energy consumption. The overhead effort of coordinating the combinatorial issues presented in other parametric methods is significantly reduced using scripts. This approach of using scripts to implement individual energy efficiency measures was well suited for the analysis since it had many different options for each building and climate being considered.

Originally, we expected that scripts developed for each measure would be sequentially called to apply additional measures, but ultimately due to the logic needed to select measures that saved the most energy, a single large script was made that combined the scripts that implemented each individual measure. The single script grew very large and complicated. An effort to refactor the script to make it more modular was considered but never implemented due to time pressures. More planning for the architecture of the script earlier in the project may also have avoided the need for refactoring.

LIBRARY FOR IDF MODIFICATION

The Python library used, called Eppy (Philip 2015), had already been partially developed as part of an EnergyPlus interface prior to the beginning of the project. During the project, the author of the Eppy library continued its development which was used to implement each of the thirty measures modeled. The Eppy library has been released under an open source license and is available at:

https://github.com/santoshphilip/eppy

Its documentation is available here:

http://pythonhosted.org/eppy/

Further development has continued even after the study was complete. Eppy is described as:

"Eppy is a scripting language for E+ idf files, and E+ output files. Eppy is written in the programming language Python. As a result it takes full advantage of the rich data structure and idioms that are available in python. You can programmatically navigate, search, and modify E+ idf files using eppy. The power of using a scripting language allows you to do the following:

Make a large number of changes in an idf file with a few lines of eppy code.

Use conditions and filters when making changes to an idf file

Make changes to multiple idf files.

Read data from the output files of an E+ simulation run.

Based to the results of an E+ simulation run, generate the input file for the next simulation run." (Philip 2015)

As an example of one of the simplest scripts which implements a measure to reduce the lighting power of exterior lights (parking lot, walkways, and entrances):

```
theIdf = IDF(oldFileName)
extLts = theIdf.idfobjects['EXTERIOR:LIGHTS']
multiplier = 0.2210
for extLt in extLts:
baselineLevel = float(extLt.Design_Level)
    extLt.Design_Level = baselineLevel * multiplier
theIdf.saveas(newFileName)
```

Those who are familiar with both Python and EnergyPlus can probably read this simple example script and know what it does.

The first line reads a file that contains the existing EnergyPlus input file for the building, the second line extracts a list of the Exterior:Lights objects from that file. Each exterior lights object is accessed in turn with the line starting with FOR. For each exterior lighting object the original Design Level, which is one of the fields for that EnergyPlus object, is multiplied by 0.221 to a revised level. The last line shows that the whole file is resaved using a new file name.

Here is the IDF object before the script:

```
Exterior:Lights,
   Exterior_Lights_a,                    !- Name
Exterior_schedule_a,                     !- Schedule Name !-
   623.03,     AstronomicalClock,        Design Level !- Control
General;                                 Option
                                             !- EndUse Subcategory
```

Here is the IDF object after the script:

```
Exterior:Lights,
   Exterior_Lights_a,                    !- Name
Exterior_schedule_a,                     !- Schedule Name !-
   137.68963,                            Design Level !- Control
AstronomicalClock,  General;             Option
                                             !- EndUse Subcategory
```

While this is the simplest script developed for a measure in the project one can see how the same process could be applied to make any change to an EnergyPlus input file that was needed.

Eppy also has the capability to construct EnergyPlus supply and demand loops by specifying a list within a list of names, such as this example *(Philip 2015)*:

```
loopnm = "p_loop"
sloop = ['sb0', ['sb1', 'sb2', 'sb3'], 'sb4']
dloop = ['db0', ['db1', 'db2', 'db3'], 'db4']  hvacbuilder.makeplantloop(idf,
loopnm, sloop, dloop)
```

After this is done, a HVAC plant loop is created for EnergyPlus with pipe objects for each item in the list and all the required branches, splitters, and mixers are created. After that, for each pipe and duct, a normal EnergyPlus object can easily be substituted using other Eppy functions. Eppy can also replace a branch with a different branch allowing for significant changes to the system configuration. In addition, since each field of an object can be represented as an item in a normal Python list (when using the Eppy "obj" function), the normal ways of manipulating lists in Python can be applied to manipulate field values which is especially useful for EnergyPlus objects like splitters, mixers, and branchlists that contain lists of names.

The Eppy library proved to be extremely capable and a good match for the national study performed. Since it is based on Python, it is a great match for anyone familiar with Python needing to work with EnergyPlus files. Eppy contains low level functionality, such as directly changing objects and fields, as well has higher level functionality, such as construction of loops. To get the most out of EnergyPlus both levels are necessary. At times, the requirement to manipulate so many different fields in various objects became time consuming when the overall goal seemed simple. This is a fault of EnergyPlus having so many inputs and the support of complex topologies. It is hoped that ways to further hide the many complexities of EnergyPlus can be incorporated into future scripting libraries.

WORKFLOWS

Two different workflows took place during the analysis: individual measure modeling and combining the measures. For each individual chosen measure, research was conducted to determine the "maxtech" variation of the measure and identify any options that should be examined. After several sources of literature related to a maxtech measure were reviewed, an individual prototype building model was manually changed to model the measure. EnergyPlus was employed almost as a testbed for a series of laboratory experiments, with small changes made and the energy savings examined until the case with the greatest energy savings was found. After that was working, an individual Python script using the Eppy library was developed that would transform a subset of the baseline files to new files implementing the measure using the parameters that resulted in the greatest energy savings. The script would reproduce the manual changes using a generic approach that could work on other building models. When a measure had multiple options, multiple files were generated for each baseline file. First the script was tried with just one prototype building and one climate until the results were reflective of the measure and errors were minimized, after that more prototype buildings were tried, still in one climate, continuing to refine the Python script. Finally, the measure was modeled in all prototype buildings for one climate (usually Chicago) and all climates for a single prototype building (usually OfficeMedium). Each measure script was developed independently and called EnergyPlus directly to run the simulations.

A separate script gathered the individual results from the XML tabular output files and collected them all into CSV files that were examined using a spreadsheet program. Error files were individually examined, sometime concatenated together first, to determine if problems existed. Since the different prototype building files contained very different HVAC systems, measures that added or modified HVAC frequently would work for some and not others, refinements to the scripts would be made to automatically detect the type of HVAC system so that the appropriate changes could be implemented. While the goal was to develop scripts that could be used on any EnergyPlus input file with any HVAC configuration, it became clear that for HVAC related measures, they would probably be closely tied to the various configurations that appeared in the prototype buildings and would likely fail for other configurations. In a few places scripts had to be extremely specific and look for objects with certain names to take action on rather than more generic approaches. An option to the individual measure modeling approach that might have been more efficient would have been to incrementally add measures from the start rather than have them all based on original prototype files.

The second workflow used during the project was employed when combining all thirty individual measure scripts (allMeasures.py). The individual measures needed to be exercised in order and selected if they reduced energy consumption since the overall goal was very low energy consumption building models. The scripts were combined into a single file and logic was added to perform the simulations in a fixed order and compare the extracted results with the previous iteration. If the site energy was reduced that measure was retained, otherwise the previous state of the input file was used for the next measure. For measures with multiple options, each option was compared and the one with the lowest energy consumption was used. To do this, the script incorporated modifying the input files using Eppy, extracting results from the XML tabular output file, and the logic to make selections.

When combining the measure scripts, multiple measures that modified the same objects were identified and reworked to ensure that changes were not overwriting changes from previous measures. The main workflow at this point was to run simulations across each prototype building and usually all climates for each measure in order; examine the errors and warnings that resulted as well as the impact on energy consumption; and update the scripts for problems found. This was usually done using the QuickSim approach (see below) using distributed computers. Even with many computers, the simulations usually took two or three days, so a weekly update cycle emerged where simulations were started for a weekend and problems were fixed from that cycle over the following week. Sometimes this update cycle stretched to every two weeks if many problems were uncovered. Scripts to collect and filter the error and warning messages from EnergyPlus were developed to help identify the problems as quickly as possible. The fact that some errors are only reported by EnergyPlus after others were fixed meant that the expectation of being on the last cycle were often wrong. The cycle of examining errors, updating the script, and running simulations was repeated many times over many months.

At the same time as the overall script was being developed, updated files that reflected a newer version of ASHRAE Standard 90.1 as well as an updated version of EnergyPlus were incorporated into the project. This added significant complications and greatly increased the number of error fixing iterations.

The process of selecting the measures prior to modeling them probably did not result in the best selections.

Instead, iterating between measure selection and modeling might have resulted in greater energy reduction. During each iteration, by examining the end use energy consumption by building prototype, measures could be selected that focused on the largest reductions possible for the next iteration.

RESULT EXTRACTION

Python scripting was also used for extracting results from the EnergyPlus tabular output files (epXML2CSV.py). By using the XML output option in EnergyPlus (see the OutputControl:Table:Style object) and the Python standard library ElementTree object, values were extracted for each simulation and stored in comma separated value files that could easily be read by other scripts. Each row of results in the CSV file represented a specific simulation. This approach is different from what is included in Eppy which can extract numbers directly from the HTML output file that EnergyPlus generates.

Two different CSV files were generated for each building and city combination. The first showed every single simulation's results whether they consumed less energy or more. These were shown sequentially with each iteration of the "counter" (as described in the File Nomenclature) for every measure and every option of each measure. Also included in this CSV was a row that repeated when a selection was made of a measure (or option of a measure) that reduced the site energy usage the most. These selected rows were also shown in a separate CSV file by themselves which represented just the combination of measures resulting in the final energy consumption. So the first CSV file represented all measures and options simulated, while the second CSV file contained just those that made up the lowest energy option.

These CSV files that were collected from the simulations were then used as the basis of another Python script (makeResultTables.py) that arranged the energy consumption and other information from each file that could be used for producing tables and graphs for the report. This allowed for easy exploration of the results throughout the process of combining the measures and debugging them. For most tables and graphs used in the final report, very little manipulation was needed in a spreadsheet prior to including them.

Once a system for extracting data from output files is used, it is very easy to extract many pieces of data. On the other hand, having lots of data does not mean that it is all examined carefully. It did not take long to realize that key pieces of data for the overall energy use and end use consumption were the focus.

Using Python to extract simulation results and rearrange the results into tables to be examined was a great approach that could have been extended even further. Excel was used to reformat the tables into a form best suited for a printed report as well as to produce graphs based on the tables. Python libraries exist to create formatted tables and graphs and may have proven useful. The Python scripts used put the data into a format easily reviewable from a quality assurance perspective and that was extremely useful given the iterative nature of workflow that combined all the measures together and worked on reducing errors and problems. It might have been even better to view the tables and graphs that ultimately were going to be included in the project report at this point for further understanding of the impacts of changes being made.

DISTRIBUTED COMPUTING

In order to perform the large number of simulations, six different PC computers were utilized as well as up to twelve simultaneous virtual machines using cloud based Amazon EC2 instances (c4.large instances were used). For each computer, between two and seven simultaneous simulations were occurring in different command shell windows. A Python script (runFromFTP.py) was used to automate the distribution of individual simulations on these various computer command shells. Each command shell simulated all the measures related to a single building prototype in a single city at a time. There were 272 different combinations that needed to be distributed across all of the computer resources. An FTP site was used as a central point for distributing the input and Python files and collecting the results of each simulation. The results were transmitted as a single compressed file for each building and city combination containing the IDF, ERR, EIO, HTML, XML and extracted values for each measure and measure option. Since simulating all the measures for a given building and city took at least an hour, it was not necessary to have a sophisticated system to avoid collisions from multiple computers selecting the same building-city combination. Instead, a simple system of a script running on each command shell on each computer would shuffle the list of 272 building-city combinations and put a "dibs" file on the FTP site if no other computer had already called "dibs" for that specific building-city combination. This approach did not require any central management. The shuffle and dibs system was simple and avoided having an additional computer dedicated to managing the distribution of work between computers. Some prototypes for certain measures (especially the daylighting control by fixture measure) used a very large amount of memory during the process and when run on computers with limited memory either crashed or ran very slowly. Due to this, the script that performed the shuffle and dibs approach to distributing the simulations favored certain prototype buildings to be run on local computers rather than cloud based instances which had less memory. This was done by splitting the list of prototype buildings into two; one that should generally run locally and one that could be run either locally or on the cloud and shuffle each list and then combine them. An earlier version of the system explicitly put specific city and building combinations in a list to be run by each specific computer instead of using the shuffle-and-dibs method. While that level of control would be good, it also proved to be too laborious, adding time prior to starting a new set of simulations. It also sometimes meant that available resources went unused since it was difficult to predict how much simulation time each building and climate would take.

The script that ran on each computer was very simple and rarely needed to be changed. It included downloading all of the other pieces including the Python script that contained all the measures. That decision saved significant time over the initial approach of using remote desktop software to set up each individual computer each time with the necessary files and start the necessary scripts.

The measures earlier in the ordered list of measures were usually related to envelope and internal loads and generally were easier to get working during the effort to debug the combination of all measures. Because of this, “preruns” were created which were compressed files of all the simulation results up to a specific measure that were confidently working together. Creation of the prerun compressed files was based on the regular results files with files after the first failure removed. These were created using another script. The compressed prerun files were put in a directory on the FTP site for all the computers to access. The way the scripts were written, these prerun files, if available, were downloaded first and the portions of the measures that they represented were not simulated since those results were already present. This greatly reduced the duplicative simulations that might have occurred and thus the amount of time to resimulate the remaining measures during the debugging process.

If files were missing, that almost always indicated a problem, sometimes a problem with the scripting system and sometimes with a particular building prototype and measure. When EnergyPlus crashes, it is common for not all output files to be generated. Making sure that all files expected are actually generated is a first step to ensuring that the system was working. Also, errors in the Python scripts that performed the distribution of simulations across computers were especially problematic so it made sense to make them as robust as possible with additional error checks.

The time to run EnergyPlus on physical computers varied significantly but the speed for using various different types of instances on the cloud did not vary as much. They were all fairly quick and the many flavors of instances available are probably more necessary for different web based duties than running numerical software. This meant that using many lower cost cloud instances was the most cost effective strategy. In addition, older physical computers may not be as useful as one might expect, because run times for the same simulations were often much longer than current generation computers or the cloud based instances.

A script to check on the status of the FTP site provided a quick update on the status of which city and building combinations were complete, in-progress, or still waiting as well as which computers were done. Another way of getting notification was an automated email generated when each command shell had no more city-buildings left to work on. At one point, developing a dashboard was considered to check the status of the project but the script that summarized the status and reported on a console was adequate. The information flow to the FTP site did not include the exact simulation that was currently running on each computer, just which computer was running each building and city combination. Additional information on exactly which measure was being simulated would have been helpful many times.

Upon consideration of the approach used, a few improvements could have been made. Since simulation times varied so much between particular prototype buildings, climate and measure combinations, it would have been good to automatically store the run times as a way to automatically optimize the usage of resources for the next iteration. Often one or two local or cloud based command shells were running many hours after others had finished. Optimally, it should be possible to balance the resources so that they finish up nearly at the same time so to maximize the utilization of each computer. In addition, only one overall analysis could be running at the same time. It would have been easy if at the start, the FTP site and shuffle-and-dibs script had been set up to run multiple separate analyses. There were times that a fix was found to one problem that could have been tested while simulations continued on a different test. The system used which only allows a single analysis at a time, meant that all simulations had to finish for that analysis and be removed from the FTP site before the next analysis started. Finally, while eventually it was figured out how to get a Windows-based cloud computer instance to automatically start two command shells and a Python script in each upon booting, this is not as simple as it probably would have been using Linux based cloud computer instances.

QUICKSIM

One technique that was used to significantly reduce the time to evaluate if the simulations for a measure worked properly was to use the QuickSim approach of simulating only the 4th, 17th, 30th, and 43rd week of each year rather than the entire year. This approach increases the simulation speed by a factor of four. An analysis by Athalye (2010) related to COMcheck showed that percent differences for simple changes were less than 1% when compared to full annual simulation results. The QuickSim approach is also being used by California Title 24 software program called CBECC-Com. The approach generates the same errors as an annual simulation so it was an expedient method of finding and fixing errors. The QuickSim approach was not used for any of the final simulations but only for detecting and diagnosing problems. Given that QuickSim was used only as a debugging tool, it would be interesting to continue to explore how much faster EnergyPlus simulations could be and generate the same warnings and errors. EnergyPlus has many controls for the length of timestep and degree of accuracy that impact the simulation speed and could be explored to make this debugging approach even faster. It is important that final simulations should always be using full year simulations and normal configurations so that any additional problems from that can be fixed.

ERROR GATHERING

As any user of EnergyPlus knows, the error and warning messages that can be generated can sometimes be quite numerous. Given that errors from over 14,000 simulations needed to be examined during each cycle to understand how to fix them, several scripts were developed to aid in digesting that information. The first (epErrIncProc.py) processed all the .err files for a specific building and city combination. It would look incrementally at each step comparing the types of errors found in in the current step to those found in the previous step. Since EnergyPlus does not uniquely number the error messages, the routine attempts to identify each type of error message by creating a "signature" for that error message. The signature is the text of the error message without the specific object names or values included. The object names can be removed because they are indicated in all uppercase letters. After a signature for each error message is generated it is counted for the current and previous file. A text file is generated for each measure added and the errors that occur with counts for each error in the current and previous step. This allows quick examination of when during the series of measures that new errors and warning are being added or when the frequency of the error increases suddenly. This approach results in 272 different ErrSummary.txt files, one for each building and climate combination.

Since the approach described above still results in 272 different files, a second level of scripting is used (createErrSummaries.py) to further summarize. The script sorted the error messages by the measureID but it also groups error messages that contained common strings such as: "CheckWarmupConvergence", "FindRootSimpleController", or "CheckForRunawayPlantTemps" were sorted by the measure ID. This grouping and sorting allowed the quick identification of which measures were still generating errors and what those errors were. It also quickly identified when a measure was failing for only certain building prototypes or failing in general.

One area that EnergyPlus could be improved is related to warnings and errors. Each unique warning, severe, or fatal error message should have a unique number that identifies the error message that does not change between releases. In addition, the messages should not be repeated since they do not add information on how to fix the problem. In general, the error messages should make it clear how to fix the problem and that should be the perspective of why they exist. Generic messages about convergence and iterations are difficult for users to fix and instead should be replaced with specific messages perhaps requiring a system that diagnoses the input file problems and recommend specific solutions. In addition, some warning messages can be ignored and others should not be. Still others can be ignored if they do not occur very often. The warning messages should specifically state when they can be ignored and allow the option to not include them.

One option that was considered but never implemented was randomly choosing a city for each iteration when running simulations across all buildings. This might have revealed the edge cases sooner that were caused by extreme conditions that occurred in only certain climates and their impacts on specific measures.

FILE NOMENCLATURE

The format for all IDF file names and thus used on the HTML, XML, ERR and other files as well was:

```
<bldg>-<city>-<counter>-<measureID>-<optionID>.idf
```

For example:

```
RetailStandalone-Duluth-0013-EO04_RfInsul-020.idf
```

This allowed a clear understanding of the measure that was added to each file. The counter is a four digit number starting from 0001 and incrementing for each simulation for a given building and city. The counter turned out to be a valuable device for understanding the order that the simulations were run but also simply because files were often shown sorted which put them into the order that the measures were attempted.

The flaw of the file naming scheme was the lack of the scenario number. Scenarios were rerunning the analysis with some zone load level measures missing in order to understand the incremental impact on later measures. The scenario number with all measures was zero and the five other scenarios were numbered 1 to 5 and incrementally removed more and more measures from the set of simulations. The various scenarios were kept in different directories, but without the scenario number in the file name, it was difficult to verify that a particular file belonged to a particular scenario. The counter would be smaller for a particular measure in later scenarios since fewer simulations would have occurred but that was not an obvious way to differentiate between files. In general, it would have been better to start off with the scenario number as part of the file nomenclature.

Because measures (or options) were only selected if they decreased energy consumption, the file nomenclature could not show which measures were actually being included in later simulations. To keep track of which measures were included in the file, the Python script kept a history as part of the name of the building in the EnergyPlus Building object. For example, the building name shown below indicated the last few measures applied to the OfficeMedium file for some specific climate:

```
Medium Office>XL01>IE03>IL06>DA02>EO04-020
```

Prior to the EO04-020 (double roof insulation) was DA02 a daylighting measure, and IL06 the interior lighting reduction measure. Unfortunately the name of a field like the Building Name is limited in EnergyPlus so only the last few measures that contributed to that file can be stored this way, but by looking at earlier files the complete history of which measures have been added to a file can be determined. Automatically created CSV files already showed the output results for each selected measure in order of the selection, but this allowed debugging of problems with specific files to be done more quickly since the history of measures was easy to follow. An alternative approach may have been better, which was to use Output:PreprocessorMessage to carry along the entire history of the measures applied to a file. This object is intended for preprocessors to add error messages to the normal error file but can simply hold a bunch of lines of text. This would not have had the limit as the single field of the Building Name and would have had the benefit of automatically showing up in the error file.

CONCLUSION

The combination of scripting tools such as Python and Eppy along with building energy modeling tools such as EnergyPlus provides a powerful way to conduct complicated analyses involving hundreds or thousands of simulations. The approach of a script doing all the modifications to the input files means that the analysis can be based on a fixed set of original files and can be reproduced and modified easily. Unfortunately, the complexity of HVAC configuration descriptions in EnergyPlus meant that some scripts became very complex in order to have special rules for different original configurations. This paper presented many details of how such a study was performed and included discussion of problems found, lessons learned, and possible solutions.

ACKNOWLEDGMENT

ASHRAE was the primary funding source of this work. For his invaluable work in leading the development of the open-source Eppy library which was used for much of the project, I would like to express my sincere gratitude to Santosh Philip.

REFERENCES

Athalye, Rahul 2010. QuickSim Analysis Report. Althalye, R.A. et.al. Pacific Northwest National Laboratory. PNNL-19661.

EnergyPlus 2015. U.S. Department of Energy, Energy Efficiency and Renewable Energy, Office of BuildingTechnologies. http://www.energyplus.net/

Glazer, Jason 2016. ASHRAE 1651-RP Final Report. Development of Maximum Technically Achievable Energy Targets for Commercial Buildings - Ultra-Low Energy Use Building Set. GARD Analytics, Inc, and ASHRAE, Atlanta, GA.

Halverson, Mark 2014. ANSI/ASHRAE/IES Standard 90.1-2013 Determination of Energy Savings: Quantitative Analysis. M. Halverson, R. Athalye, M. Rosenberg, Y. Xie, W. Wang, R. Hart, J. Zhang, S. Goel, V. Mendon. Pacific Northwest National Laboratory. PNNL-23479.

Philip, Santosh 2015. Eppy scripting langues for EnergyPlus idf files and EnergyPlus output files. https://github.com/santoshphilip/eppy, https://pypi.python.org/pypi/eppy/, http://pythonhosted.org//eppy/. Loisos + Ubbelohde. Alameda, CA.