2018 Building Performance Analysis Conference and
SimBuild co-organized by ASHRAE and IBPSA-USA
Chicago, IL
September 26-28, 2018

SCRIPTING FRAMEWORKS FOR ENHANCING ENERGYPLUS MODELING PRODUCTIVITY

Amir Roth[1], Jamie Bull[2], Scott Criswell[3], Peter Ellis[4], Jason Glazer[5], David Goldwasser[6], Neal Kruis[4], Andrew Parker[6], Santosh Philip[7], David Reddy[8]

[1]US Department of Energy, Washington, DC, US, [2]oCo Carbon, London, UK, [3]SAC Software, Portland, OR, US, [4]Big Ladder Software, Denver, CO, US, [5]GARD Analytics, Arlington Heights, IL, US , [6]National Renewable Energy Laboratory, Golden, CO, US [7] Loiso +Ubbelohde, Alameda, CA, US, [8]360 Analytics, Seattle, WA, US

ABSTRACT

Task automation is a great productivity enhancer in any workflow. In building energy modeling, scripting has emerged as a powerful tool for automating both standard tasks such as generating a code-minimum baseline model from a model of a proposed or actual building and for workflow customization. This paper presents four scripting frameworks for EnergyPlus. Eppy makes EnergyPlus input objects accessible as Python objects, allowing them to be manipulated programmatically. Modelkit is an embedded Ruby library that allows modelers to create parameterized templates that instantiate different specific EnergyPlus input files depending on parameter values. OpenStudio Measures provide programmatic access to the OpenStudio object model; that model is translated into an EnergyPlus input file. CBECC uses a custom scripting/rule engine that operates on the Standards Data Dictionary (SDD) building model schema; OpenStudio is used to translate SDD files to EnergyPlus input files. The paper discusses the philosophy behind each system, its architecture and core capabilities, and uses simple modeling tasks to show how each system is used in practice.

INTRODUCTION

Addressing the energy and environmental challenges of the 21st century will require an all hands approach and commitment. Whole-building energy modeling (BEM) has the potential to make a significant contribution to this effort, supporting dramatically improved design and operation of homes and commercial buildings which currently account for 40% of energy use and 70% of electricity use in the US.

If the BEM enterprise is to scale and realize its potential, it must overcome several challenges. One of these is modeler productivity. Let's do some quick and dirty math. About one million new homes are built in the US every year along with about two billion square feet of commercial space. Roughly the same amount undergoes a significant addition or retrofit. Assume that each new or retrofitted home or 1000 square feet of commercial space requires five hours of modeler time. That's 30,000,000 hours of modeler time per year or 15,000 full time modelers. The bad news is that the number of modelers in the US is much lower than 15,000. The worse news is that the modeling time assumptions are probably low and that much modeling time is spent on tasks like performance documentation for code-compliance or green certificates that don't contribute directly to improved building performance. Along with recruiting and training additional modelers, a productivity multiplier is needed that dramatically reduces the effort associated with performance-documentation activities but also enhances analyses that contribute directly to building performance. That accelerant is automation. The adage "anything that *can* be done by a computer *should* be done by a computer" is aging in an unanticipated way as computers can now do things like play chess, drive cars, and perform surgery. But its original intent "computers should do repetitive, mechanistic, non-creative tasks and leave original, analytical, and creative work to humans" has only started to play out in the BEM world.

This paper presents four scripting frameworks that are actively being used to automate different modeling tasks and workflows with EnergyPlus (EnergyPlus, 2018). Scripting is lightweight computer programming using dynamic languages like Python and Ruby that abstract away many details associated with traditional, compiled programming languages like FORTRAN and C++. Scripts are more concise and readable than compiled language source code, and more portable and shareable. And although they are slower than compiled programs, they are still many orders of magnitude faster than humans performing the same tasks manually.

```
Construction,
    WallCons-1,  !- Name
    RG01,        !- Outside Layer
    BR01,        !- Layer 2
    IN46,        !- Layer 3
    WD01;        !- Layer 4

BuildingSurface:Detailed,
   Wall-North-1, !- Name
   Wall,         !- Surface type
   WallCons-1,   !- Construction name
   Zone-North-1, !- Zone name
   Outdoors,     !- Outside boundary
   ...
```

Figure 1. EnergyPlus construction and surface objects.

```
# Set window construction on walls
# with >= 40% WWR
model.getSurfaces.each do |surf|
  # Skip walls with WWR < 40%
  next if surf.windowToWallRatio < 0.4
  # For all windows on this wall
  surf.subSurfaces.each do |window|
    # Set construction
    window.setConstruction(high_wwr_glass)
  end
end
```

Figure 2. Ruby script that sets window constructions

Figure 1 shows two EnergyPlus objects: a construction object named `WallCons-1` and a Surface object that references the construction object, i.e., the construction of the surface is `WallCons-1`. Green text beginning with `!-` is "comments", inline documentation that is not part of the code. Figure 2 shows a Ruby script that sets the construction for all windows on walls with a high window-to-wall ratio (WWR) to a certain construction object. In Ruby, comments begin with #. With inline documentation and cursory knowledge of programming keywords like `do`, `next` and `if`—keywords are built-in commands, in Ruby `do` iterates over a collection of elements, `next` skips to the next iteration and `if` tests a conditional—the script is mostly readable even by non-programmers. And while the time it takes to write and test this script may not be recovered on the first project on which it is used, it will be on subsequent projects. Just as one does not need to be a mechanic in order to drive or to understand TCP/IP or SMTP in order to send email, a modeler does not need to know how to write scripts in order to use scripts written by others. A single script guru can empower an entire organization. And the "bite-size" nature of scripts makes them more likely to be shared broadly. A modeler is more likely to give away a script that took a day to develop than a compiled program that took a year. OpenStudio's BCL (Building Component Library, 2018) contains over 300 Measures, many of them created and posted by users.

Scripting is extremely useful for developing and analyzing individual models; not only does it reduce effort, it also reduces human error, which degrades the quality of modeling-based decisions and often leads to additional modeling effort later. Scripting is absolutely essential for conducting large-scale analyses, either with individual specific models (design and retrofit optimization, model input calibration, uncertainty analysis) or with collection of prototype models (deemed savings calculations, demand forecasting, energy-efficiency program planning). The cloud has made large scale BEM accessible and economically viable for organizations and projects of almost any size. Scripting is the key to systematically and efficiently generating the model variants that put the cloud to work.

EnergyPlus ships with two pre-processing utilities that support limited "embedded" scripting, i.e., scripting from with the EnergyPlus input IDF file itself (EnergyPlus Input/Output Reference 2018). ParametricPreprocessor interprets EnergyPlus parametric objects to generate a group of input file variants. Parametric specifications can vary field values, remove entire objects, and use conditional logic for greater flexibility. EP-Macro is a more general-purpose file-level pre-processor with capabilities for file inclusion, conditional object inclusion, and a greater variety of expressions. Both facilities have found use among practitioners, as well as among client applications, EP-Macro's use in jEPlus (jEPlus, 2018) is one example.

The four scripting frameworks described in this paper go beyond these simpler facilities. Some use general-purpose scripting languages with a full complement of programming features and libraries. Some split scripting from the EnergyPlus input file, improving portability and allowing scripts to operate on existing files. Some do both. The frameworks have overlapping use cases but are tailored for different purposes. Some operate directly on EnergyPlus input files while others operate on other data models which are then translated into EnergyPlus input files. Some are lightweight with gentle learning curves, while others have a greater range of capabilities, but also steeper ramps and more attendant "luggage".

- Eppy (EnergyPlus PYthon) is a library that makes EnergyPlus input objects available as Python objects, allowing modelers to manipulate them directly in code. Eppy targets energy conservation measure (ECM) evaluation and large scale analysis.
- Modelkit is a template system and library that works directly within EnergyPlus input files. Written in Ruby, Modelkit targets model development, ECM evaluation, and parametric analysis.
- OpenStudio Measures are Ruby scripts that operate on OpenStudio models, which are translated to EnergyPlus input files. Measures target model development, ECM evaluation, and large scale analysis.
- CBECC includes a custom scripting language operating on models described in the SDD schema, and currently translates SDD CBECC-Com models into EnergyPlus input files (via OpenStudio), and CBECC-Res models into CSE input files.

Each of the next four sections describes a scripting system, its design philosophy and intended set of applications, and gives several examples of how the system is used in practice.

EPPY

Eppy provides low-level programmatic access to EnergyPlus inputs in Python, a popular language with high-level features that support "one liners" for many common programming idioms, e.g., extracting all objects that match a specification from a list (Eppy, 2018). Eppy parses EnergyPlus IDF files and creates a Python object for each corresponding EnergyPlus object. Users can programmatically manipulate object fields, delete objects, and create new objects. Eppy writes out the updated Python objects to a new EnergyPlus IDF file.

```
# Extract all surface objects
Surfkey = 'BUILDINGSURFACE:DETAILED'
surfs = idf.idfobjects[surfkey]
# Extract all walls
walls = [s for s in surfs if s.tilt == 90]
# Extract all north-facing walls
nwalls = [w for w in walls if w.azimuth == 0]
# For each north facing wall
for nwall in nwalls:
   # Set construction
   nwall.Construction_Name = newconstruction
```

Figure 3. Eppy script that sets wall constructions

Figure 3 shows the construction "search-and-replace" idiom in eppy. Surface objects are extracted from the EnergyPlus object list and north-facing walls extracted from the surface list using common Python "one-liners".

In Python, object fields are accessed by name. In EnergyPlus input files, fields are accessed by their position in the object. EnergyPlus fields do have names, but these are given in the EnergyPlus input schema file (the Input Data Dictionary or IDD), which describes the legal structure of EnergyPlus input files. Eppy uses the IDD to create names for EnergyPlus object fields. Eppy's reliance on the IDD file allows it to handle EnergyPlus version updates easily. In general, eppy code can read any EnergyPlus input file, in any version.

In addition to fields defined in the IDD file, eppy also adds some fields for generally-useful calculated properties. In the example above, `tilt` and `azimuth` are not EnergyPlus Surface properties. They are eppy Surface object fields that are calculated from the EnergyPlus Surface object's vertex list. Eppy also includes helper functions for viewing HVAC loops graphically and for "walking" through HVAC loops by stepping forward or backward to the next or previous component. Finally, eppy includes functions for running EnergyPlus simulations—including running simulations in parallel—and reading simulation results. These building blocks allow eppy to support iterative analyses and optimizations with results from one simulation used to determine changes in subsequent simulations.

Example Uses

Eppy is designed for direct low-level manipulation of EnergyPlus input files. The most extensive use of eppy to date is the ASHRAE RP-1651 on "max tech" commercial buildings (Glazer 2015, 2016). Here eppy was used to evaluate 30 different ECMs on 16 commercial building prototype models across 17 climate zones. The ECMs modeled ranged in complexity from changing lighting power to hybrid ventilation. Other complicated ECMs included external light shelves, underfloor air distribution, chilled beam, radiant heating and cooling with DOAS, ground-source heat pumps, and daylighting control by fixture.

There are several public repositories of eppy "recipes" including geomeppy which targets geometry generation, export, visualization and modification (geomeppy, 2018). Several projects extend eppy with higher-level functionality. weppy is a web interface for viewing the structure of EnergyPlus input files (weppy, 2016). While the eppy libraries do not yet include high-level functions that build geometry or entire HVAC systems, the basic scripting power of eppy allows these to be implemented over time. Some of these may developed as part of Py90dot1, an initiative to generate ASHRAE90.1 baseline models using eppy (py90dot1, 2018).

MODELKIT

Modelkit is general-purpose scripting framework for building energy modeling. Originally developed under the name Params (Ellis, 2015), the framework was focused on parametric analysis and automatic generation of large batches of EnergyPlus simulations. The parametric framework was recently extended with additional "toolkits" for unit conversions, psychrometrics, and batch job queueing, creating a more general purpose platform and inspiring a name change to Modelkit. The core objective of Modelkit is to provide a lightweight, fast, and flexible framework for modeling process automation.

Modelkit::Parametrics—the artist formerly known as Params—is the framework's most mature module. It includes a templating system and a comprehensive library of EnergyPlus templates. Templates consist of standard EnergyPlus input file syntax "marked up" or parameterized with dynamic content in the Ruby scripting language. Figure 4 shows the Modelkit template for an EnergyPlus Material object. The EnergyPlus Material object itself is easily recognized at the bottom of the snippet. Text enclosed in <% %> pairs is embedded Ruby code. In this case, the object has two parameters, reflectance and emittance. A third variable, absorptance, is calculated from reflectance.

```
# Template parameters
<%#INITIALIZE
parameter "reflectance", :default=>0.3
parameter "emittance", :default=>0.85
%>

# Internal variables
<% absorptance = 1.0 - reflectance %>

#EnergyPlus object template
Material,
  Roof Membrane,        !- Name {}
  VeryRough,            !- Roughness {}
  0.0095,               !- Thickness {m}
  0.16,                 !- Conductivity {W/m-K}
  1121.29,              !- Density {kg/m3}
  1460.0,              !- Specific Heat {J/kg-K}
  <%= emittance %>,     !- Thermal Absorptance
  <%= absorptance %>,   !- Solar Absorptance
  <%= absorptance %>;   !- Visible Absorptance
```

Figure 4. Modelkit template for EnergyPlus Material

The Modelkit EnergyPlus template library includes ready-to-use templates for space loads, zone HVAC equipment, central HVAC systems, and more. Templates are semantic units that organize related input objects into meaningful high-level data structures. They include the configuration details of how input objects are connected together to achieve a desired simulation behavior. In this way, templates provide a level of abstraction and encapsulate a significant body of knowledge with respect to modeling best practices in EnergyPlus. The library templates in Modelkit are parameterized to include "baked-in" ECMs. In addition to the efficiency measures that can be applied by changing a single parameter (e.g., lighting power density, fan efficiency, or chiller COP), there are parameters that act as "switches" to make larger changes to the model in order to simulate measures such as fan/pump type (constant vs. variable speed), boiler type (non-condensing vs. condensing), condenser type (air-cooled vs. water-cooled), daylighting controls, and even different HVAC system types. These ECMs have been developed and refined over numerous modeling projects.

Because they are mostly made up of IDF objects, templates can be readily modified and extended by modelers and other non-programmers. Templates can also be easily combined with non-template IDF files. The templating system was developed for EnergyPlus but is generic enough to be used with any tool that uses text-based input files. It has also been applied to OpenStudio, CBECC-Res, and CBECC-Com.

In Modelkit, templates are hierarchical with top level templates instantiating lower-level templates. Low-level templates implement individual objects or related groups of objects, while higher level templates select between lower-level templates and instantiate them. The top level template is the "root" template. This hierarchical structure allows Modelkit to use control logic in the root template to make significant high-level changes to the final model, including selecting and configuring entire HVAC systems.

In addition to templates, Modelkit includes a module that specifically facilitates working with the EnergyPlus simulation engine. A Ruby library (borrowed from Euclid/Legacy OpenStudio) parses EnergyPlus inputs and dictionary files and allows objects be created, deleted, modified, and queried programmatically similar to eppy—except in Ruby rather than python. The module can also run batches of simulations, post-process results, and aggregate results across large batches.

Modelers typically interact with the Modelkit framework using the command-line interface to execute specific tasks such as "composing" a template, running an EnergyPlus simulation, or post-processing an EnergyPlus SQL output file. An Excel spreadsheet tool is available to provide a simple user interface for the most common commands. Modelers can also write their own Ruby scripts that access the classes and methods of the framework directly as a packaged Ruby library (known as a "gem").

Example Uses

Modelkit can be used for parametric analysis of individual buildings or larger, utility-scale analyses of prototype models simulated across multiple climate zones and multiple ECMs. The framework serves as the building modeling backend for the SMPL/NZP/District Zero tool (Case et al. 2014; Ellis 2016) developed by the US Army Corps of Engineers and Big Ladder. Created as a decision-making aid for Army planners and energy managers, this platform provides a web-based graphical interface that allows users to model energy-saving scenarios for hundreds of buildings at Department of Defense (DoD) bases and installations.

OPENSTUDIO MEASURES

OpenStudio is a software development kit (SDK) for EnergyPlus. OpenStudio's data model closely resembles that of EnergyPlus but is not identical. Some EnergyPlus features are not accessible through the OpenStudio model (OpenStudio, 2018). OpenStudio also adds some constructs—primarily Space and SpaceType—that are not native to EnergyPlus, but that are useful in many applications. The OpenStudio data model is accessible programmatically via an Application Programming Interface (API), a set of functions that manipulate the data model at various levels, from low-level updates of individual fields in individual objects to high-level geometry transformations and HVAC system creation and configuration. The higher-level functions abstract away a lot of esoteric EnergyPlus objects. For instance, OpenStudio AirLoopHVAC functions automatically create, delete, connect and disconnect EnergyPlus Node, Branch, ZoneMixer, ZoneSplitter and various List objects—saving time and error, especially in large models. Critically, the OpenStudio API also has methods to access EnergyPlus simulation outputs. OpenStudio interacts with EnergyPlus by translating its building data model to an IDF file prior to simulation and accessing EnergyPlus outputs afterwards.

OpenStudio supports scripting by providing "bindings" that allow the API to be invoked from dynamic languages such as Python, JavaScript, C#, and Ruby. OpenStudio has a special relationship with Ruby scripts in that it bundles in a Ruby interpreter that can execute these scripts. This arrangement is similar to the one in Microsoft Excel, which bundles a Visual Basic interpreter and can execute VB scripts on spreadsheets. Some OpenStudio scripts called Measures have a particular structure which allows OpenStudio to understand where and how Measures can be invoked and to automatically combine and chain Measures according to specification to automate large scale analysis. Measures can be invoked from any OpenStudio-based application, including the minimalist OpenStudio command-line interface (CLI). Measures may be shared on the Building Component Library (BCL). The BCL allows Measures to be shared selectively, e.g., within an organization but not with the public.

Example Uses

Many of the Measures posted on the BCL implement ECMs. These range from simple search-and-replace transformations such as changing lighting power densities to more complex ones such as changing-out entire HVAC systems. A visually compelling Measure is the "AEDG Daylighting Package" Measure which performs a coordinated set of transformations aimed at improving daylight harvesting: configuring windows and skylights, changing glazing types, adding shading, placing lighting sensors, and programming lighting control strategies. A "before-and-after" of this Measure is shown in Figure 7 below.

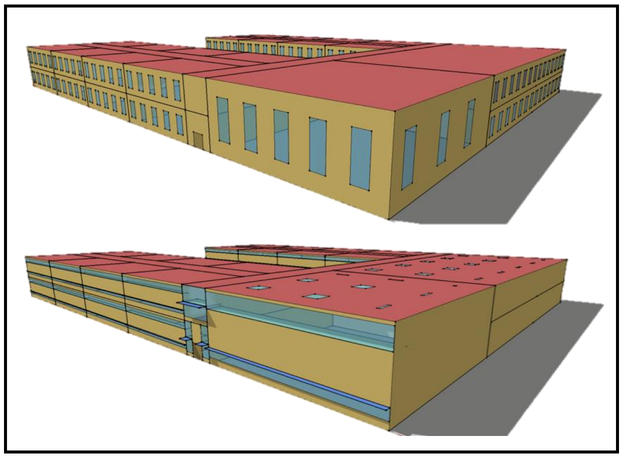

Figure 5. OpenStudio Daylighting Package Measure

With access to both simulation inputs and results, Measures can also be used to create reports and visualizations, perform diagnostic and quality assurance checks, and create mixed analyses that connect EnergyPlus with other engines such as Radiance or GLHEPro. The standard reports and graphics in the OpenStudio Application are created by a Measure.

The most complex Measure written to date automatically creates an ASHRAE 90.1-2013 Appendix G baseline (aka LEED baseline) model from the proposed model, including difficult aspects such as HVAC fuel type and system creation, autosizing of HVAC equipment and efficiency application, daylight area calculations and sensor placement. Users only need to pick a type for each space from a list (corridor, restroom, closed office, etc.); everything else is truly automatic. The "Create Baseline Building" Measure is understandably and necessarily complex, but highlights the power and flexibility of OpenStudio scripting as well as some aspects of the data model including the Space and SpaceType constructs.

The real power of Measures manifests when they are combined to create large-scale analyses. Measure-based analyses have been used to optimize individual building designs, to identify cost-effective retrofit candidates in a portfolio of buildings, and to calculate deemed savings for ECMs using prototype models.

CBECC

CBECC-Com and CBECC-Res are the California Energy Commission's (CEC) software tools used to conduct performance-based T24 energy code compliance analysis (CBECC-Com Non-Residential Compliance Software, 2018). At the heart of each tool is the ruleset, which contains the data model definition, as well as a series of commands, or rules, that default user model values, quality-check inputs, transform the user's model into a series of simulation models according to prescribed rules, and report results. The data model used by CBECC, the Standards Data Dictionary (SDD), was developed to bridge the gap between real-world building design, and the information needed for simulation tools (Pepetone 2017). The structure and application of rules is flexible, but optimized for developing proposed and baseline models for code and above-code compliance applications. Rules are written in a custom language that has matured over 20 years of development, with the principal goal of creating a software development environment engineers and modelers can use with little previous programming knowledge. An important feature of the ruleset is the organization of reference data in tables and libraries, as it might be found in energy codes or other references. CBECC is accessible as a stand-alone desktop application, and to third-party BEM programs via an API. CBECC-Res and -Com both utilize the same rules processing machinery, but have their own independent rulesets, and generate simulation input files for different simulation engines; CSE (California Simulation Engine, 2018) for CBECC-Res analysis, and EnergyPlus (via OpenStudio) for CBECC-Com.

Custom `RULELISTs` allow energy modelers to manipulate the proposed design CBECC model according to their own rules. Below is an example of a simple `RULELIST` containing one rule that sets the construction assembly object reference for all new, north facing walls of conditioned spaces. The name of the `RULELIST` is declared, followed by rules that will be executed when the `RULELIST` is evaluated. Each rule has an `object:property` target that will be evaluated on, and the subsequent logic defines the criteria for executing a command.

```
ConsAssm   "WallSFU0.064"
   CompatibleSurfType = "ExteriorWall"
   ExtRoughness = "MediumRough"
   ExtSolAbs = 0.7 ExtThrmlAbs = 0.9
   ExtVisAbs = 0.8 IntSolAbs = 0.7
   IntThrmlAbs = 0.9  IntVisAbs = 0.8
   SpecMthd = "Layers"
   MatRef = ( "Stucco - 3/8 in.",
              "Gypsum Board - 5/8 in.",
              "Compliance Insulation R13.50",
              "Gypsum Board - 5/8 in." )
```

Figure 6. CBECC construction assembly object.

```
RULELIST "AssignNorthWallConst" 1 0 1 0
;*********************************************
"Assign construction to new north facing
ExtWalls on conditioned spaces"
ExtWall:ConsAssmRef = {
  if( Status = "New" .AND.
      (RealAz > 315 .OR. RealAz <= 45) .AND.
      Spc:CondgType != "Unconditioned" )
  then
    ; Assign library construction to wall
    RuleLibrary( ConsAssm, "WallSFU0.064" )
  else
    ; Otherwise, skip wall
    UNCHANGED
  endif
  }
END
```

Figure 7. CBECC rule that sets wall constructions.

In the example, the `ConsAssm` object named "`WallSFU0.064`" in the ruleset library is assigned to the property "`ConsAssmRef`" of the "`ExtWall`" object. Each rule is evaluated for all the target objects in the model, but in this case, only assigns the `ConsAssm` object to the walls that meet the rule logic. `RULELISTs` can contain multiple rules with different `object:property` targets, and logic within rules can reference any model property that is available via valid data model relationships.

CBECC's custom scripting language has most of the features of general-purpose languages including simple data types like integers and strings, compound types like objects and arrays, arithmetic and logical expressions, and conditional execution. All of these appear in the example above. The CBECC language is missing two common general-purpose language features, loops and user-defined functions. However, for the target application of applying transformations to building models these are not absolutely necessary and CBECC gets around them in different ways. In lieu of loops, the `RULELIST` construct is understood to be evaluated over all matching `object:property` instances and the CBECC interpreter performs this specific form of looping automatically. In lieu of user-defined functions, rules can invoke other `RULESET`s. CBECC also has over 160 built-in functions that assist in the creation and modification of building models. Some functions are highly specific to BEM, such as calculating the daylit zone areas and control positions based on 3D space boundaries and window locations and sizes.

CBECC's use of the SDD data model and a custom language has both advantages and disadvantages. Learning a new data model is always a time investment, but the SDD has an intuitive structure with properties familiar to modelers, designers, and code officials, as opposed to the esoteric structure and names used in simulation engines. The description of spaces, thermal zones, and building geometry should be familiar to gbMXL users (gbXML, 2018). The SDD HVAC model is a hybrid of real-world inputs and EnergyPlus simulation properties, while simplifying the description of HVAC systems by using parent, child, and reference relationships. To develop `RULELIST`s in CBECC, users must have some familiarity with the SDD and learn the syntax used in CBECC rules. Compared to other scripting languages, modelers will likely find the syntax more human readable. The lack of function calls and looping makes program control "flat" and more visually and logically traceable. No additional software, bindings or modules are needed to use custom `RULELIST`s.

In addition to changes like those described in the example above, the CBECC-Com interface allows users to: assign custom defaults based on object type or name, create and assign HVAC systems to individual zones or groups of zones, assign proposed model capacities/efficiencies based on look-ups from custom tables, Scale window areas and/or replacement of fenestration or surface construction performance based on orientation or other logic, and create custom reports of model data, exported to text files or a UI message box.

Custom `RULELIST`s are a useful extension to the CBECC modeling workflow, especially if a third-party software that has integrated the CBECC API does not support scripting, or the proposed compliance model requires bulk changes. Overall, the SDD, custom `RULELIST`s, and publicly available compliance rulesets offer a robust set of tools for a variety of applications.

Example Uses

Since 2014, CBECC-Com and -Res have been extensively used, either integrated into commercial BEM software or in a freely available standalone form, to determine performance-based compliance with the 2013, 2016, and (in the near future) 2019 T24 Energy Efficiency Standards (Alatorre, 2015) (Ferris, 2015). The current CBECC-Com framework supports multiple nonresidential compliance scenarios (new complete, additions and alterations, core and shell), building applications (residential, multifamily, laboratory, data center, commercial kitchens), and energy efficiency technologies (energy storage, VRF, DOAS, chilled beams). In addition to T24 compliance applications, CBECC-Com offers 90.1-2010 Appendix G compliance analysis using the same proposed model data. CBECC also includes a batch processing feature to automate bulk testing and parametric analysis.

CONCLUSION

The scripting frameworks presented in this paper exhibit a range of philosophies, capabilities, and target applications. From direct low-level access to EnergyPlus input objects to higher-level data models that require a translation step to EnergyPlus and may provide incomplete coverage of its features. From use of general purpose scripting languages and execution environments to domain-specific languages and environments that specifically target energy models and energy modeling processes. From a focus on compliance and standard-based modeling tasks to support for large-scale simulation. From open-source licensing to … open-source licensing. The latter is important as it ensures the frameworks will remain dynamic, with each continuing to evolve, add functionality and supporting content, and attract users and new use cases. A future paper that revisits these frameworks could look substantially different and include several new frameworks.

The scripting framework that is right for you and your organization depends on your target application, your background and experience, and, perhaps, whichever one you try first! More important than the choice of a specific scripting system is starting to leverage scripting of any kind in your workflow. Scripting is a powerful productivity enhancer. Learning how to use scripts written by others requires no programming experience and minimal general computer wherewithal. With the number and variety of freely available scripts already large and growing, even this minimal engagement can yield significant productivity gains. Learning to modify scripts written by others takes a little more effort but can be done incrementally, gives significantly more freedom and flexibility, and is fun. Learning to write scripts from scratch is probably not necessary for the great majority of modelers, especially given the many scripts already available. However, it will give you infinite power—at least over your energy modeling workflow—should you decide to undertake it. Whichever approach and system you choose, happy scripting.

ACKNOWLEDGMENTS

The authors thank the funding agencies that have financially supported these platforms including the U.S. Department of Energy, the U.S. Department of Defense, the California Energy Commission, and ASHRAE; the many who have used and contributed to these tools; and the reviewers for their constructive comments.

REFERENCES

About Measures [Website] 2018. Retrieved from http://nrel.github.io/OpenStudio-user-documentation/getting_started/about_measures/

Alatorre, M., Bozorgchami, P. et al. 2015. 2016 Nonresidential Alternative Calculation Method Reference Manual. California Energy Commission, Building Standards Office. CEC-400-2015-025-SF. http://www.energy.ca.gov/2015publications/CEC-400-2015-025/CEC-400-2015-025-CMF.pdf

Building Component Library [Website] 2018. Retrieved from http://bcl.nrel.gov/

California Simulation Engine [Computer Software] 2018. Retrieved from https://cse-sim.github.io/cse/

Case, M., Liesen, R. et al. 2014. "A Computational Framework for Low Energy Community Analysis and Optimization", NY-14-011, ASHRAE Transactions, Volume 120, Part 1. Atlanta, GA.

Brook, M., Criswell, S. 2012. The BEE Software Collaborative: An Open Source, Rule-Based Architecture for Building Energy Efficiency. ACEEE conference, Aug. 2012, Pacific Grove, CA.

CBECC-Com Non-Residential Compliance Software [Computer Software] 2018. Retrieved from http://bees.archenergy.com/

Criswell, S., Glazer, J., et al. 2014. Functional Requirements for ACM Standards Compliance Engine Software, Version 1.0. San Francisco, CA: NORESCO.

Ellis, P. 2015. "Parametric Modeling with Templates and Scripting", Presentation at 2015 ASHRAE Energy Modeling Conference, Sep.-Oct. 2015, Atlanta, GA.

Ellis, P. 2016. "A Parametric Tool for Community-Scale Modeling", Proceedings of SimBuild 2016, ASHRAE and IBPSA-USA National Conference, Aug. 2016, Salt Lake City, UT.

EnergyPlus [Website] 2018. Retrieved from http://energyplus.net/

EnergyPlus Input/Output Reference [Website] 2018. Retrieved from https://bigladdersoftware.com/epx/docs/8-8/input-output-reference/

Eppy [Computer Software] 2018. Retrieved from http://www.github.com/santoshphilip/eppy/

Eppy documentation [Website] 2018. Retrieved from http://eppy.readthedocs.io/en/latest/

Ferris, T., Froess, L., et al. 2015. 2016 Residential Alternative Calculation Method Reference Manual. California Energy Commission, Building Standards Office. CEC-400-2015-024-CMF-REV3. http://www.energy.ca.gov/2015publications/CEC-400-2015-024/CEC-400-2015-024-CMF-REV3.pdf

gbXML [Website] 2018. Retrieved from http://www.gbxml.org/

Geomeppy [Computer software] 2018. Retrieved from https://github.com/jamiebull1/geomeppy

Goldwasser, D., Macumber, D., et al. 2016. "The Life Cycle of an OpenStudio Measure: Development, Testing, Distribution, and Application", Proceedings of the ASHRAE and IBPSA-USA SimBuild and Building Performance Modeling Conference, Aug. 8-12, 2016, Salt Lake City, UT.

Glazer, J. 2015. ASHRAE 1651-RP "Development of Maximum Technically Achievable Energy Targets for Commercial Buildings Ultra-Low Energy Use Building Set." ASHRAE final report. Atlanta, GA.

Glazer, J. 2016. "Using Python and Eppy for a Large National Simulation Study." ASHRAE/IPBSA-USA Building Simulation Conference, Aug. 10-12, 2016, Salt Lake City, UT.

Hale, E., Macumber, D., et al. 2012. "Scripted Building Energy Modeling and Analysis." Proceedings of IBPSA-USA SimBuild, Aug. 1-3, 2012. Madison, WI.

jEPlus [Computer Software] 2018. Retrieved from http://www.jeplus.org/wiki/doku.php

OpenStudio [Computer Software] 2018. Retrieved from https://openstudio.net/

Pepetone, D. et al. 2017. "2016 Title 24 (CBECC-Com) Nonresidential Compliance Engine Standards Data Dictionary" (SDD spreadsheet).

Params [Computer Software] 2018. Retrieved from https://bigladdersoftware.com/projects/params/

Parker, A., Benne, K., et al. 2014. "Parametric Analysis Tool for Building Energy Design Workflows: Application to a Utility Design Assistance Incentive Program", Proceedings of the 2014 ACEEE Summer Study on Energy Efficiency in Buildings, Aug. 2014, Pacific Grove, CA.

Py90dot1 [Computer Software] (2018). Retrieved from https://github.com/pyenergyplus/py90dot1

Roth, A., Goldwasser, D. et al. 2016. "There's a Measure For That!", Energy and Buildings, Volume 117, Apr. 2016.

Weppy [Computer Software] 2016. Retrieved from https://github.com/santoshphilip/weppy, hosted at https://weppy.herokuapp.com