THE LIFE CYCLE OF AN OPENSTUDIO MEASURE:
DEVELOPMENT, TESTING, DISTRIBUTION, AND APPLICATION

David Goldwasser¹, Daniel Macumber¹, Andrew Parker¹,
Edwin Lee¹, Rob Guglielmetti¹, Larry Brackney¹

¹Commercial Buildings Group, National Renewable Energy Laboratory, Golden, CO

ABSTRACT
An OpenStudio Measure is a script that can manipulate an OpenStudio model and associated data to apply energy conservation measures (ECMs), run supplemental simulations, or visualize simulation results. The OpenStudio software development kit (SDK) and accessibility of the Ruby scripting language makes measure authorship accessible to both software developers and energy modelers. This paper discusses the life cycle of an OpenStudio Measure from development, testing, and distribution, to application.

INTRODUCTION
The OpenStudio SDK acts as an object-oriented abstraction for building energy modeling (BEM) that simplifies the effort of creating models and related software applications. The SDK has been designed to target both the EnergyPlus and Radiance simulation engines, and supports a number of analysis use cases including new construction design, model calibration, retrofit assessment, and large-scale technology potential analyses. Example applications built with the SDK include the Trimble SketchUp Plug-In for geometry creation, the OpenStudio Application often used for specifying detailed simulation inputs, and the Parametric Analysis Tool (PAT) for comparative analysis of design alternatives. As with many BEM tools, the Plug-In and Application may be used to manually create and modify building models, however the real power of the SDK becomes more apparent when full procedures for creating and modifying models are captured in a prescribed form of Ruby script. Such scripts are referred to generally as OpenStudio Measures, and are so named because they are most frequently used to represent the application of an ECM to an arbitrary building model, however their functionality need not be limited to ECM application.

OpenStudio Measures (referred to simply as “measures” throughout this paper) are written in the Ruby scripting language, and are structured in such a way as to be used in any OpenStudio-based application via the SDK’s application program interface (API). Standardization of a measure’s API calls is used to identify key input parameters and outputs such that measures can easily be “plugged” into a variety of workflows and graphical user interfaces (GUIs). The API exposes the full OpenStudio object model and supporting methods for use within a measure, facilitating efficient and accessible manipulation of model structure, data, and simulation results that may be used to apply specific building retrofits, standardize reporting, or automate analysis operations.

Additionally, measures may be “chained together” to create sophisticated and repeatable workflows for individual or portfolio-scale analysis. Lastly, the online Building Component Library (BCL)¹ and associated API linkages provide a seamless method of delivering measure content to a growing ecosystem of OpenStudio-based tools. OpenStudio Measures have emerged as a means of encapsulating expert knowledge and best practice that is being embraced by a growing number of design professionals and tool developers to improve the cost and confidence of modeling outcomes.

A number of third party applications use the OpenStudio API with free-form scripts that are not structured as formal measures. Such scripts can automate modeling tasks just as measures do, but may not adhere to API as required by all applications. This prevents them from being used in a seamless, “drag and drop” fashion as formal measures can be.

¹https://bcl.nrel.gov/
Standardizing on the OpenStudio Measure structure also enables cost and effort sharing across a larger community. As an example, a number of utilities have implemented quality assurance/quality control (QAQC) measures as part of their new construction incentive programs. By implementing QAQC checks as measures, all of the participating utilities and their consulting firms can easily share their process automation. Further, practitioners not working within these utility programs may also enjoy the benefits of the QAQC measures, increasing their confidence in modeled results.

Having defined the concept and value of measures, we now focus on the process of creating and using them. Perhaps the first order of business is to determine the need for a formal measure. Is there a modeling task or workflow step that is commonly performed and would benefit from automation? Perhaps a design project requires some degree of parametric analysis to identify optimal design parameters? Is there a need to package a potentially complex modeling approach so that it can be shared with a broader community? These are the types of situations that often warrant the creation of a measure. A discussion of the various stages in the life cycle of a measure is the subject of the rest of this paper. Figure 1 introduces the notion of a measure life cycle based on its intended use and audience.

Measure Development
Measures are perhaps the most focused examples of OpenStudio applications, making very specific use of the API so that they can be used as broadly and seamlessly as possible². If the use case for a measure is to automate a one-time tedious task for a single model, the development process can be very simple, however such measures will be only narrowly applicable. In general, measures that will be used repeatedly and by multiple modelers benefit from a more thoughtful and structured development process. The authors recommend beginning the process by creating a design document that describes:

- The objective for the measure and use cases,
- The kinds of models it is expected to operate on,
- Any parameters that should be exposed as input arguments for the user,
- The general approach to implementation,
- Any informational, warning, and error messages that the measure should report out, and
- A summary of test cases that will be used to verify that the measure works as intended.

None of the recommended design document content requires deep knowledge of OpenStudio or programming experience – only subject matter expertise. The authors have found that engaging stakeholders in the design document development process is key to creating quality outcomes, and is an excellent stepping stone towards developing proficiency in writing and using measures. Once complete, the design document is a blueprint for identifying which elements of the API will be leveraged, followed by coding and evaluation of the measure. The authors also recommend that the document be maintained alongside the measure, throughout its useful life, as a transparent and useful reference describing design intent and assumption.

² [http://nrel.github.io/OpenStudio-user-documentation/getting_started/about_measures](http://nrel.github.io/OpenStudio-user-documentation/getting_started/about_measures)
WRITING THE MEASURE

When creating a new measure within the OpenStudio Application or PAT, a functional template is created. The template allows quick editing of measure metadata used by the BCL, automatically creating the correct measure directory and file structure along with a simple “hello world” measure.rb file as a starting point. New measure developers are best served by first duplicating an existing, proven measure, then making small, incremental changes that extend functionality for periodic testing.

A typical measure.rb file is split into an arguments section, responsible for parsing the measure’s inputs, and a run method, containing the logic that defines the measure’s functionality. Arguments may be numbers, strings, choice lists, or boolean variables. Choice list enumerations may be hard coded or dynamic, querying of available model objects or the BCL. Adhering to the argument method structure and data types enables integration of the measure in any OpenStudio application, presenting the user with a seamless means of specifying arguments.

A typical run section of a measure is shown in Figure 2. The run method is responsible for any model articulation, modification, or report generation that the measure performs. Measures have access to all OpenStudio model objects and methods. Measures may also access all EnergyPlus objects, including objects that may not be directly supported by the OpenStudio SDK. In addition to OpenStudio SDK and EnergyPlus documentation, the OpenStudio Measure Writing Guide provides specific guidance and examples for writing measures. Lastly, measures may access native Ruby methods. There are many web resources for Ruby guidance.

![Figure 2 - Run section of new OpenStudio measure](image)

MEASURE TESTING

Measure authors intent on making their products available for broader use owe their users some level of testing rigor. Whether the author’s audiences are a few practitioners within their own firm or stakeholders providing regulatory oversight of utility incentive programs, measures are and will be used to drive business decision-making. A well-written design document will describe a range of applicable use cases along with test conditions and expected results. This section discusses unit and integration testing in the context of OpenStudio Measures.

Unit tests are typically performed around a specific section of software. Unit test are considered a minimum standard for verifying that a measure performs as expected despite changes to the underlying SDK or measure code itself. One or more unit tests should be distributed with each measure. Different unit tests may test different combinations of argument values or input models. The most basic test assures that the measure runs without errors, but more advanced tests inspect the

---

3 https://www.youtube.com/watch?v=1heU3VYny64
4 http://openstudio.net/developers and http://energyplus.net/documentation
5 http://nrel.github.io/OpenStudio-user-documentation/reference/measure_writing_guide
6 https://www.ruby-lang.org
output data, log messages and the resulting model to see if they match expectations. These tests can be written ahead of, during, or after measure development.

Integration tests go beyond assessment of what a measure did to the model to encompass other aspects of performance. A common example is comparison of annual, monthly, or time series simulation results to reference data sets. Other measures, which generate reports or apply quality assurance heuristics to models, may compare their output (typically html files) over a range of test cases. Integration tests may exercise multiple measures applied in different combinations or on different baseline models. While integration tests can be used for any measure, they are particularly valuable for measure collections that are intended to work in concert.

Modern software development processes rely on automated testing and “continuous integration” frameworks to constantly assess performance in the face of changes to compilers, software dependencies, etc. As more organizations rely on EnergyPlus and OpenStudio to drive business decisions, the Department of Energy and National Laboratories have deployed rigorous testing frameworks and processes for these projects. Quality assurance of measures is equally important, and work is underway to automate testing of BCL measure content. In the near future, measure authors will be alerted when unit tests on the BCL happen to fail. This automation is intended to produce confidence among stakeholders, ensuring that software updates do not unexpectedly impact the energy savings performance of ECMs or compromise automated quality checking that utility incentive programs rely on.

MEASURE DISTRIBUTION

If a measure was written for individual use or for use by a small private group, then the development and distribution locations could be one and the same – e.g., local network drive, cloud based drive, or (preferably) a version control system. More commonly, measures are intended for broader distribution to a community of stakeholders. The BCL was intended to facilitate sharing and seamless distribution of this content. While users can directly search for measure content using a web browser, the BCL was intended to integrate with OpenStudio-based applications seamlessly via API. This provides a superior user experience that allows measure content to be made available directly where it is needed – within analysis tools. Measures are also overloaded with metadata that helps classify and describe a particular measure. The BCL search engine and API leverage this metadata to assist users in quickly locating the exact content they are looking for. A measure taxonomy organizes content into major and minor sub-categories as shown in Figure 3. If a modeler were looking for a measure to modify outdoor air ventilation, they would first look under “HVAC” and then under “Ventilation.” Some measures may reasonably span multiple taxonomy categories. In this case, the measure author should choose a single category that is most appropriate. Additional measure metadata such as the modeler description can be viewed prior to downloading a measure, and provides additional clarity beyond what might be inferred from the measure’s title and taxonomy category.

In addition, the OpenStudio API makes use of measure metadata to check for OpenStudio version compatibility. This prevents users from downloading and using measures that may not be supported by their version of OpenStudio. Because the OpenStudio development team goes to great lengths to maintain the stability of the API, measures developed with old versions of OpenStudio will generally run properly in newer versions. Because measures are distributed outside of the OpenStudio installation, updates can be made at any time without waiting for a quarterly release of OpenStudio. This has proven particularly useful for large organizations such as universities that may have a limited number of times a year they can update software on their networks. A measure author may update their measure at any time to correct issues or add new functionality, and make it available to the community via the BCL within minutes. OpenStudio applications use the BCL API to notify users that updated measure content is available. The user may then elect to download the updated content or use their current version. It is worth noting that while the BCL does track and maintain measure versions, the example OpenStudio applications currently do not facilitate downloading specific, older versions of measures to support replication of old analyses.

While this section has focused on public sharing of measure content via the BCL, the authors recognize that many firms consider their approach to modeling ECMs, visualization, or reporting to provide them with a competitive advantage. To that end, the BCL allows for content to be uploaded and maintained as private within a group. This enables firms to distribute content within their organization, and make it public only if they choose to. The example OpenStudio applications do not currently support the latest version of the BCL.
API that facilitates private content search. As such, groups using the BCL to manage private content must use the web interface for download.

BEM desktop software, or even lightweight web or mobile applications.

The OpenStudio Application illustrates two general approaches for applying measures. In the most basic approach, a modeler specifies a single measure that runs immediately. In the case of an ECM, the model is modified and the user can elect to apply additional measures, manually modify the model, or run a simulation. This is demonstrated with the “Apply Measures Now” dialog in the OpenStudio application (Figure 4).

![Figure 4 – Apply Measures Now Dialog](image)

A second approach is for one or more measures to be applied to a prescribed baseline model prior to simulation as part of a more complete run-time workflow. This approach does not alter the working model; instead it records a set of run-time instructions telling OpenStudio what measure to run and with what user argument values. This is demonstrated in the “Measures” tab of the OpenStudio application (Figure 5). In both of these cases the user has an opportunity to set arguments that the measure author has exposed. While ECMs can be applied in this manner, a more common use is to apply measures that prescribe utility rates or perform additional reporting or visualizations on the model.

Parametric analysis is perhaps a more compelling application of measures. The OpenStudio PAT example illustrates this use case by applying multiple measures to a single baseline model. In this scenario one or more of the arguments from a measure become variables that may change from one design alternative to another. In the current version of PAT the user specifies the measures to be considered in an analysis along with the various argument values (Figure 6).

![Figure 3 - BCL Measure Taxonomy](image)
These measures are then manually combined to form the design alternatives that are to be simulated. Creating an interface that runs measures need not require the development of an elaborate GUI. The OpenStudio Analysis Spreadsheet is an example of this. Since this application runs exclusively on Amazon EC2 using a server image preconfigured with EnergyPlus, OpenStudio, and Radiance; the user need not even install OpenStudio on their local computer. Here, the interface used to specify the analysis is an Excel spreadsheet (Figure 7).

A small Ruby script parses the spreadsheet contents, formulating an analysis problem that is transmitted to the primary OpenStudio server. The server initializes an appropriate number of worker nodes, parallelizes the simulations, manages the requested analysis, and acts as an interactive results visualization interface via any web browser. The spreadsheet is capable of defining a number of different analyses including optimizations, sensitivity analysis, and sampling problems using a number of user selectable algorithms. Problems that are commonly solved with the OpenStudio spreadsheet include model calibration, retrofit or new construction design optimization, parametric sensitivity analysis, and uncertainty quantification. This functionality is currently being added to PAT and will be available in OpenStudio version 2.0 to be released in September of 2016.

Honeybee, a plugin for Grasshopper and Rhino that helps explore and evaluate environmental performance for buildings, is an example of a third party application directly using OpenStudio measures within its own interface. Figure 8 shows a Radiance Measure being used for daylighting analysis. This was accomplished even though the primary scripting language for Rhino is not Ruby. While Honeybee already has a robust set of features, access to OpenStudio Measures expands the functionality to include the full collection of measures on BCL. Grasshopper modelers can for example utilize the DOE Prototype building measure to create an initial model, and know that it will be the same as a DOE prototype model made in any other OpenStudio application.

Figure 7 – OpenStudio Analysis Spreadsheet

7 http://nrel.github.io/OpenStudio-user-documentation/tutorials/large_scale_analysis/
CONCLUSIONS
The last two years have seen the development of measure content that, for example:

- Implements ASHRAE Advanced Energy Design Guide recommendations for Offices and K-12 Schools,
- Performs rigorous daylighting analysis for a variety of envelope configurations and shading devices,
- Generates entire models with only a few parameters (e.g., building type, vintage, and climate zone),
- Converts proposed models to 90.1 compliant baselines,
- Creates interactive psychrometric charts and 3D geometry viewers,
- Imports data from legacy eQuest models, and
- Believe it or not, extracts model schedule data from in situ Nest thermostats.

As adoption of OpenStudio Measures grows, it becomes increasingly important that developers focus on those best practices that make it easy for users to find and use the content they need, that the content works as expected across applications, and that the results may be trusted. In addition to describing some of the benefits of measures, this paper has attempted to identify some key considerations and resources that can make the difference between a measure people will want to use, and those that fall short of expectations.

Some key recommendations include:

- Developing a measure design prior to coding,
- Making use of existing measures and the measure authoring guide as resources,
- Approaching unit and integration testing seriously and with rigor, and
- Being thoughtful in BCL publishing and metadata considerations such as naming, taxonomy, and functional descriptions.

In addition, it’s important to note that the pace of OpenStudio measure development is rapid and accelerating. New developers are encouraged to look at already published content and keep abreast of development in progress before writing a new measure. Developers can save time, avoiding duplicative effort, and users will benefit by eliminating confusion associated with redundant content.

This is an exciting time to be involved in energy efficiency and modeling. Advances in software technology are creating new opportunities for more cost effective analysis that the industry has sorely needed. The authors believe that OpenStudio measures are an important “common currency” that can propel these innovations further and faster; enabling collaborative development of content and new tools that help practitioners identify deeper energy savings more quickly than ever before.

---

Figure 8 - OpenStudio Measures in Honeybee (Image courtesy of Mostapha Sadeghipour)

Measures under development are documented whenever possible at https://docs.google.com/spreadsheet/ccc?key=0AhlCiA4LtwiaGRdE1BDUJZDQ01iLVBmUFJpCHEjWU10ZEEUSop-drive_web-gid=0
ACKNOWLEDGMENT

The authors thank Amir Roth, and Jason Hartke of the DOE Building Technologies Program for supporting the development of EnergyPlus and OpenStudio. We also thank the EnergyPlus and OpenStudio development teams and our partners at LBNL, ORNL, PNNL, and private sector companies. Thanks also to the growing community of users and measure content developers.

REFERENCES

1 NREL, Building Component Library, retrieved March 29 from https://bcl.nrel.gov/
2 NREL, About Measures, retrieved March 29 from http://nrel.github.io/OpenStudio-user-documentation/getting_started/about_measures/
3 NREL, OpenStudio 0.11.0 Parametric Analysis Tool (Writing Energy Conservation Measures), retrieved March 29 from https://www.youtube.com/watch?v=1hcU3VYny64
6 ruby-lang.org, Ruby website, retrieved March 29, 2016 from https://www.ruby-lang.org
8 NREL, OpenStudio Measures Roadmap”, retrieved March 29, 2016 from https://docs.google.com/spreadsheets/d/1QUP6Nz1X6gdOcNEIBuxByONHOz5m_tTpr-llp2avp0/edit - gid=0