


# AUTOBRICK: A SYSTEM FOR END-TO-END AUTOMATION OF BUILDING POINT LABELS TO BRICK TURTLE FILES

PREPRINT, COMPILED MARCH 7, 2021

Devanshu Desai <sup>1\*</sup> and Advitya Gemawat <sup>2†</sup>

<sup>1</sup>Halicoglu Data Science Institute, University of California at San Diego

<sup>2</sup>Halicoglu Data Science Institute, University of California at San Diego

## ABSTRACT

BRICK is a schema for representing various building equipment, including but not limited to, HVAC air handling units and carbon dioxide sensors in different rooms. While the schema is a clear step up over the current state-of-the-art, its potential is severely hindered because it is not backwards compatible. This means that converting CSV files storing building data to a BRICK-compatible data format is a cumbersome and imperfect process as different systems use different conventions to denote the same systems. This conversion usually required human involvement until now. AUTOBRICK is a software tool that automates this conversion with minimal human intervention and provides an order of magnitude greater speed up (90x) over the current state of the art.

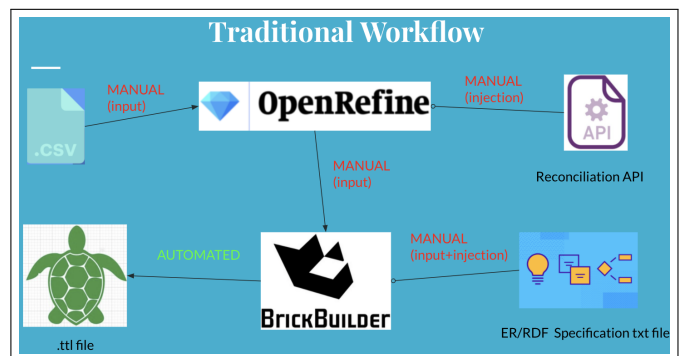
## 1 INTRODUCTION

Today, several commercial buildings have sought to leverage the power of cyber-physical systems for optimal energy consumption, planned facility maintenance, machine learning (ML) applications, etc., to augment user experience. However, their data collection / ETL methodologies don't conform to a uniform schema to enable buildings to interact with each other or streamline the process of analytics and ML at scale. The US building industry suffers a loss of \$1.5B / year due to the lack of interoperable warehousing between different buildings [1]. Brick offers a comprehensive schema to represent arbitrary buildings metadata, build programmatic platforms for commercial buildings, improve interoperability between different buildings, and pave the way for well-defined ways of analyses related to planned maintenance and machine learning, etc. [1].

Converting arbitrary metadata schema to Brick (i.e., 'Brickification') is a slow process bogged down by manual interventions into partially-automated processes and grunt work. This process involves custom data transformations, object-relationship definitions, and manually plugging intermediate results into various heterogeneous frameworks to get a Turtle file parseable by Brick. A Turtle file is a data format that expresses data in a RDF model which can be used to store graph-like info for relationships between different entities, in this case, building sub-systems and sensors. Different kinds of building point label (metadata string) formats that sabotage the notion of a one-size fits all solution further intensify these challenges. Additionally, this is also motivated by real-life use cases such as the COVID-19 global pandemic, where the public safety regulations in indoor environments (classrooms, offices, etc.) remain a massively unsolved research question and challenge at scale.

1. an open-source Excel-like tool called OpenRefine, where users need to perform manual data transformations [2],
2. a Reconciliation API, to infer the class of specific objects in terms of Bricks vocabulary [3],
3. Entity-relationship definitions in a .txt file, to utilize in the turtle file [4], and
4. a tool called Brick Builder to convert the processed data into its turtle file [5].

As symbolized in Figure 1, the data transformations and most of the API injections and integrations need to be manually done by the user. The BrickBuilder tool is the only component to automate the last part of this workflow. We recognize the research and practical utility of the frameworks already created for use in the old Brickification workload. We don't seek to reinvent the wheel here but address automation, acceleration, and scalability improvements on top of these existing systems.



## 2 BACKGROUND

The traditional Brickification workflow involves:

Figure 1: Traditional pre-existing Brickification workflow as per the tutorial in [2]

\*correspondence: dbdesai@ucsd.edu

### 3 OUR FRAMEWORK

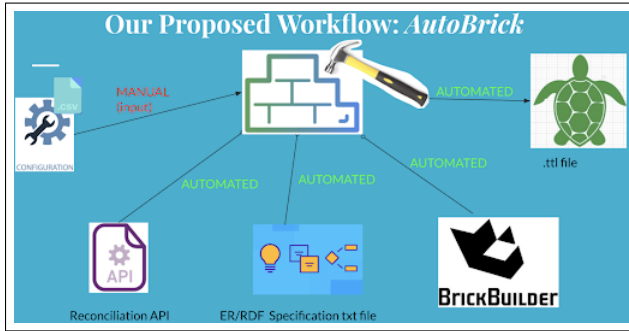


Figure 2: The proposed workflow through AUTOBRICK

AUTOBRICK is an end-to-end system for automating and accelerating the conversion of point labels into turtle files with Brick terminology, which can be visualized, queried, and utilized by the Brick programming platform. As displayed in Figure 2, the tool takes a configuration file where users can declaratively specify all details about their data and workload (related to data location, point label format details, and additional parsing details). AUTOBRICK automates all needed data transformations to pre-process the point labels and extract relevant information of various units/sensors and categories related to specific building objects. AUTOBRICK utilizes pre-written / pre-loaded standard ER/RDF specifications and existing APIs to inject and stitch them on the pre-processed data as per the original sequence of steps followed for Brickification. We recognize the research and practical utility of the frameworks already created for use in the old Brickification workload and address automation, acceleration, and scalability improvements on top of the existing implementations.

AUTOBRICK allows for flexibility in using arbitrary CSV files and chosen/specified point label formats. With the project, we give the power in the users hands for them to decide the appropriate dataset parsing/preprocessing strategies, rather than making unverified assumptions about point label structures or utilizing unreliable custom static analysis or auto-generation techniques. AUTOBRICK has already "brickified" 41 point label datasets of various buildings in UCSDs Engineering Department, reducing the need for human intervention to a matter of minutes. Initial experiments and demos have already showcased *Autobrickify* reducing 10-15 mins of manual grunt work as part of "brickifying" one dataset (as part of the tutorial in [2]) to a matter of <10 seconds, achieving orders-of-magnitude (over 90x) speed-ups and scalability!

### 4 METHODS

We use a combination of python and shell scripts to execute the workload in terms of the core logic to parse the point labels and loading and injecting pre-existing APIs at different stages of the process. To automate data transformations, we use the pandas module to load the data in memory and leverage the supplied configurations to perform data manipulations with in-built functions provided by the module to identify objects/pieces of equipment such as AHUs, Zones, VAVs, and their corresponding class types.

Utilizing shell, we write some custom scripts to clone the APIs, load them in the execution environment, and call them according to their instructions to supply the pre-processed data to their API function calls automatically. Because modules such as pandas are pre-installed in a standard python environment, the tool doesn't have any unique dependencies. It just needs to rely on the brick-builder APIs dependencies (i.e., two modules, brickschema and rdflib). The lack of dependencies further makes the tool lightweight for smooth usage. The requirements are directly taken (and hence can even be updated) from the loaded APIs themselves, making it seamless to piggyback on top of the APIs made for Brickification.

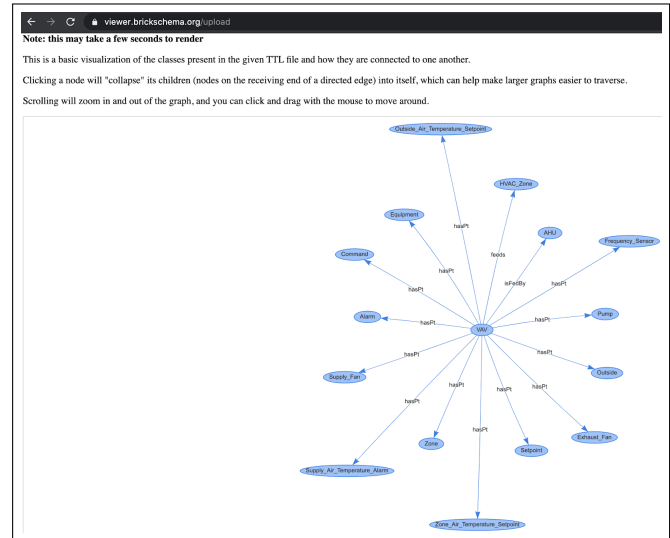


Figure 3: We can also use the Brick Viewer tool [5] to eyeball the relationship graph of the building sub-systems.

Our validation process involves both automated and manual aspects. Since we don't have any target files to compare our outputs with, we use domain experts working under our mentor. Domain experts carefully go through the generated turtle files and verify if the outputs match their expectations. The automated component involves running some automated test queries on the generated files to ensure that the graphical model can be parsed error-free, and we've showcased some of such queries in one of our demos as well.

### 5 WORK EXTENSIONS

One of the biggest challenges faced in the Brickification workflow is the predictions of Brick Classes by the Reconciliation API, which have only been around 30% accurate [2]. The Reconciliation API uses an extensive key-value pair dictionary to extract and attach relevant ontology for a sensor abbreviation in a point label, but the API isn't comprehensive. Ultimately, the turtle files need to be precisely correct for effective ETL and Analytics utilization. To mitigate this, we also formulated scripts to extract additional key-value pairs from Johnson Controls documentation to update the API and get more accurate predictions. However, this process of incorporating all key-value pairs may never be fully comprehensive and is subject to future updates. On the bright side, AUTOBRICK directly uses the latest ver-

sion of the API, automatically reflecting better results as the API improves.

The second part of augmenting the tools robustness relates to dealing with point label values with an unequal number of splits with the specified delimiters. In our current implementation, we assume the point labels split into a uniform number of elements. For point labels that split into fewer values, we consider these point-labels invalid (as we did encounter spurious point label values during our testing). Filtering out these point-labels renders the remaining values invalid. During the preprocessing stage, we drop rows with any null values. In other words, we treat point labels with the maximum number of splits possible in the dataset as the only legitimate kind of point label value for that dataset. Dealing with diverse but correct formats of point labels in a single dataset is an open research question needing collaboration with domain experts and is left to future work.

Finally, we also implemented a virtual reality front end to interact and query with the automatically generated turtle files. The environment is set up as a scene on a platform called ARENA [6]. We set up a server that listens for events and user interactions with the 3D objects in ARENA and translates those interactions into queries performed on the BRICK-compatible building information. This environment allows us to map physical spaces into equivalent virtual ones.

## 6 RESULTS AND CONCLUSION

AUTOBRICK automates and accelerates a painfully manual process, thereby empowering building management and vendors to truly realize the power of data-driven applications and advanced analytics to augment user experience in commercial buildings. AUTOBRICK also gives rise to the potential of representing building sub-systems as virtual spaces to reduce the need for engineers to physically climb into the building’s HVAC system equipment and virtually observe and interact with sensor setpoints. To truly reap the benefits of AUTOBRICK, the workflow and use case coverage need to be continually examined by domain experts and industry practitioners to ensure the tool meets their needs at all times. This paper aims to set the foundation for tangibly realizing smart buildings and adopt technological advancements moving forward.

## REFERENCES

- [1] Balaji et al. Brick: Towards a unified metadata schema for buildings. 2016.
- [2] gtfierro225. Making a brick model with openrefine brick-builder, Oct 2020. URL <https://www.youtube.com/watch?v=LKcXMvrxXzE>.
- [3] BrickSchema. Brickschema/reconciliation-api. URL <https://github.com/BrickSchema/reconciliation-api>.
- [4] Gtfierro. gtfierro/brick-builder. URL <https://github.com/gtfierro/brick-builder>.
- [5] URL <https://viewer.brickschema.org/>.
- [6] Conix-Center. conix-center/arena-core. URL <https://github.com/conix-center/ARENA-core>.