AS502502

Autodesk Forge Data Exchange APIs: Standardized Granular Data Extraction to Reduce Code Base

Robert Manna
Sr. Solutions Consultant | dRofus

James Mazza
Solutions Architect | Stantec

Learning Objectives

- Discover the value of the Autodesk Forge APIs for developing custom solutions.
- Learn how to apply Autodesk Data Exchanges to workflows that require subsets of data to be shared across multiple apps and teams.
- Evaluate the potential value in creating your own custom solution.
- Learn about the long-term value of Autodesk's move to granular cloud data and how to capitalize on that value through APIs.

Description

For several years, Stantec has maintained a software stack used to extract data from Revit software for various purposes. Most recently, this was in support of a benchmarking effort to collect standardized, normalized data from our models for reporting and analysis by project type and sector. In this scenario, users "submit" their data after performing a QA check. Autodesk Data Exchange and APIs present a method to standardize the data extraction process and give control to the end user in regards to what data is exported for downstream use. This process removes the compute from the local desktop and reduces the complexity of the internal code base. Using Microsoft Azure as a data pipeline, we move the data traffic off the corporate network (LAN, WAN, VPN, and ISP). This class will discuss our development of a solution that maximizes the Autodesk Forge Data APIs. In addition to benchmarking, we'll touch on some of the other areas where we have experimented with the tooling, including as part of our digital twin project.

Speaker(s)

Robert spent 19 years working for an Architecture & Engineering company first assisting with the initial adoption and roll-out of Revit in 2005 and moving onto larger, more strategic technology implementations and strategy. Robert transitioned to leading the development of custom Revit tools and then to analyzing and modeling data for architecture projects as well as firm-wide data analysis, modeling and transformation. Most recently Robert worked with a team developing a digital twin solution built on the Azure platform. Robert is now with dRofus as a solutions consultant where he works with key client accounts to understand their most complex challenges and develop processes and solutions to suit their needs. Robert has presented at numerous conferences and events over the years and is active with the Digital Built Environment Organization, currently serving as the Chair of the Design Technology Summit and the North American Executive Committee member.

James has spent 14 years working in AEC. His career path has spanned a broad swath of roles from 2d draughting to BIM management before transitioning to full time software development. Currently James works as a solutions architect where he designs and evaluates software solutions to serve Stantec's buildings design teams.

AUTODESK UNIVERSITY

Table of Contents

Introduction to the Course 4
Why are we interested in Forge Data Exchanges? 4
  Hypothesis 5
Where we Started 6
  Technical Details 6
So What? 8
  Data Structure 8
  Data Transformation & Flattening 9
  GraphQL 10
Developing 10
  Basics 10
  Exploratory Tooling 11

Introduction to the Course

This class is focused on two key points, the purpose and value of Forge Data Exchanges and key information you should be aware of if you want to proceed with writing your own tools and applications to take advantage of Data Exchanges. This handout is meant primarily as a supplement to the live presentation and/or recording.

Before getting into the details both abstract and technical a few words about how this course came to be and where 'we' are today.

First, Data Exchanges have been only publicly available for a relatively short period of time (April 2022), we were fortunate to have 'early' access, but that access was more on theoretical than practical side for a variety of reasons. Submitting this course involved a bit of 'risk' since we didn't entirely know how far we would get in a relatively short amount of time while also balancing our day-to-day job responsibilities.

Second, Robert (primary-speaker) changed jobs and companies about mid-way through. The author would like to acknowledge and thank both if former organization (Stantec) and his new (dRofus) for continuing to support the delivery of this course, including James' (co-speaker, Stantec) ability to participate and contribute.

Lastly the author owes enormous gratitude to his partner James, for his participation and involvement, as without it none of this would be possible.

Why are we interested in Forge Data Exchanges?

Before delving into the details of what an Exchange is, and why it is interesting, we need to better understand why Exchanges caught our attention in the first place.

Stantec and dRofus both share a common trait, an inherent interest in data for various purposes. Stantec as a professional services firm, is interested in two qualities of data:

- How can it be applied to benefit a project? This might take the form of:
    - Custom tools that can help users create or set-up their Revit models based purely on data or information about the project, or information from other existing project models.
    - Custom tools that can move data between models. For example, Stantec's own take on Revit's Copy/Monitor functionality, that instead focuses on transferring data as well as Family substitution.
- How can data from projects be leveraged to benefit the enterprise or future projects:
    - Frequently model health or performance comes up in this conversation.
    - Additionally capturing raw data from project models, means that Stantec would then be able to 'benchmark' common sizes of typical physical spaces, for example Exam Rooms, or quantities of items, ex. Number of fume hoods in a laboratory or per area.

dRofus is a software vendor whose primary application ‘dRofus’ is a database for storing, manipulating, managing, reporting and sharing Project Data or managing common data between multiple projects. Therefore, dRofus is very interested in:

- Easily getting data out of models.
- Easily pushing data into models.
- The moving of data should follow specific rules about what the authoritative source is.

None of these concepts or ideas are particularly ‘new’. Stantec’s peer organizations also undertake similar efforts and dRofus has various competitors in the marketplace (sometimes home-grown tools within organizations).

The big point here is:

Data Transfer and Exchange!!

The challenge is that Revit is a desktop application, which means that traditionally to move data in and out of Revit you must write your own application that will at a minimum interact with the desktop too, this has implications:

- you must write to the application’s APIs
    - this likely means you need to be writing C#
    - which means you need a C# developer
    - the developer will need to learn or familiarize themselves with the desktop APIs
        - Desktops APIs are often unique to the application itself
- If you want to extract (or write) data to 3rd party models (i.e., models your organization doesn’t directly manage) you must distribute your application to those third parties.

Forge Data Exchanges presented themselves as a possible avenue to avoiding the above:

- Web APIs and web technologies
- Generically structured data rather than application specific structured data
- Versioning, storage and access managed by the Autodesk platform

Hypothesis

Therefore ‘we’ in theory can focus on retrieving the data of interest and manipulating as we see fit, rather than having to handle extracting the data out of the source file/application directly. This theoretically reduces or eliminates some of the code stack that was previously required purely to interface with the desktop application. Furthermore, with Forge being an extensible, web-based platform we are potentially writing tools that can be applied to other data sources that contribute to the platform environment.

Where we Started

What are Exchanges, exactly?

- Exchanges are bundles of data that have been extracted from a model or file.
- The intent of an exchange is that the end user who 'owns' the model or file can define the contents of the Exchange.
    - In the case of Revit, initially this is accomplished by creating a 3D view and manipulating the view properties to define what is 'visible' in the view.
    - Revit exchanges however **will contain data not typically 'visible'** in a view, in particular data about 'Rooms' which many will know are not technically 'visible in a 3D view'.
    - **This has the effect of allowing a user to share only a portion of their model, furthermore, because the exchange is generic 'data' it also means that in the case of Revit a user is not directly sharing custom families, or other 'content' that perhaps they do not want to share with 3rd parties.**
- Currently Exchanges are created by:
    - Upload/publish a model with a predefined, published view to Autodesk Construction Cloud.
    - Browse to and view the target model, select the pre-defined view, and in the view, list right click to access the "Create Exchange" menu.
    - Fill out the necessary details and confirm creation of the exchange.
- Exchanges are "stored" in the ACC folder structure, which means that **ACC's permission and access control tools provide governance** over access to the Exchange Data.
- **It's important to note** that while Exchanges give the 'appearance' of a file in the ACC environment, they are a pointer to a specific collection of data stored in the underlying data repository. Beneath the covers all Exchanges are stored in a common data architecture, not as individual JSON documents or something similar.

- Currently creation of Exchanges is exclusive to the ACC environment. Autodesk is working on a Revit Desktop plugin that allows a Desktop User to Generate an Exchange from the model they have active in Revit.
    - **This has some interesting implications:**
        - In theory 3rd parties could use the same API endpoints to create their own exchanges.
        - Exchanges creation need not be limited to Revit.
        - In this scenario, the source model itself is not published publicly to ACC, but the Exchange is published to ACC for consumption.
        - Updates to Exchanges published in this way would occur from the Desktop App, not by virtue of the model being updated in ACC.

Technical Details

- Exchanges are accessed via REST API endpoints.
- Like any REST API, this means that you get 'all' the data, filtering must occur after the data has been downloaded.
    - Even on a small Exchange there is a lot of data, due in part to the construction of the payload.

- The structure of the data is deep.
- Data is normalized to Exchange ID values, not the native application IDs.
  - For you to display the data in an interface useful to end users, you'll have to construct the user readable information from the Data Schema, joined to the data as part of the Extract, Load, Transform (ELT) process
- Technically the data is constructed as a Graph, with nodes and edges, not a more typical relational structure.
  - You can convert it to relational as part of your ETL process.
  - Autodesk Documentation here.
    - **Note:** that the worked example using a "room" is not a good example in the context of Revit. Revit Rooms contain no geometry themselves; they are bounded by other geometry. The worked example assumes that the room object has geometry.
  - **For more information on Graph Databases:** we highly recommend this Pluralsight course as a good introduction to the philosophy and approach to a Graph. Don't be deterred by the fact that the course uses Neo4j.

In addition to the public REST API endpoints, Autodesk built endpoints to support the Microsoft Power Automate connector that is released for Forge Data Exchanges. Power Automate is meant to be a "low code" solution that can help 'everyday users' automate various tasks. The intent is that users could use Exchanges to gain access to Revit parameter data via a Power Automate Flow. As a low-code environment though, the experience during and the result coming from Power Automate needs to be relatively "user friendly". To meet these requirements the API endpoints for Power Automate take data transformation a few steps further than the public end-points. This alternative data structure is a little bit easier to work with than the deeper and more complex results obtained via the Public APIs, however as the Power Automate end-points are not public, they are generally not worth further consideration.

So What?

GETTING DATA IS GREAT, WHAT DO I DO NOW?

Getting data is the 'relatively easy part of this puzzle' (more discussions later), the bigger question is what do you do with all the snazzy data you just pulled in from the endpoint?

Data Structure

- As previously noted, Stantec has been exploring data extraction from Revit models for some time.
    - This includes extracting the data out to JSON, then collecting the JSON for ETL/ELT workflows to meet different requirements.
- It was a useful and interesting exercise to compare the structure designed at Stantec with the structure that Autodesk designed for Exchanges.
- While they are of course not identical matches, there are a number of similarities.
    - The biggest difference is that Autodesk, in thinking about larger scale has opted to convert all the data into a generic assembly. The source of the data is captured as part of the data schema, but the data schema itself is fairly platform and tool agnostic.
    - Stantec opted to collect data under a collection that clearly attributes the data to Revit and then keeps the data mostly in a Revit 'form' that any Revit user is likely to recognize and be familiar with.
    - In both cases parameters' user facing names are replaced with ID/s or an enumeration value that can be used as a Look-up for the actual Parameter Name.
        - Stantec opted to use the unique ID value of the Parameter in the project. This has implications for the downstream ETL/ELT and potentially scalability.
        - Stantec also stores the Parameter schema in the same JSON that contains the raw model data extracted via the Exchange.
        - The advantages are that all data is in a single data payload and in a format that expert or elite Subject Matter Experts will recognize.
        - Autodesk opts to generate a unique identifier for the parameters as part of the process for creating an exchange.
        - The actual parameter data and enumerated names is stored separately and must be fetched separately using the Data Schema API endpoint.
    - While the tree structure of the JSON varies somewhat, both data schemas eventually arrive at an object in the JSON that represents a single model object from the Revit model. That object then has all the associated parameters for that object.
        - **It is critically important to note that in both cases (Stantec & Autodesk) the structure is normalized and not flattened.**
        - This means that an object you arrive at, could be one of several possible 'item types':
            1. The top-level definition of the Family (in Revit terms), this includes basic data like the Family Name, Category, Family Parameters (which are limited).

2. The definition of a single Family Type, related to the top-level Family definition. This would store the Type parameters (properties) of the object.
3. The definition of an instance (or 'occurrence') of a specific Family Type in the larger model. Including the Instance parameters (properties) that are unique to that specific individual type instance.

- **Note:** This is where an understanding of how Revit structures its data can be useful. While knowledge of the Revit API is not necessary, even though the data has been abstracted out of the Revit format, the developer must still know how to navigate and transform the data.

Data Transformation & Flattening

- Exchanges are not 'small' payloads of data. While undoubtedly in the greater world of technology much larger JSONs are being moved. One of the very small samples we worked with was nearly 5,000 lines, which represented nominally 50 unique instances of objects from the source model.
- We experimented with transforming/manipulating the data payload in Power Query (M).
- From an ELT perspective our goal was table(s) of objects by Category, By Family, By Type, By Instance with 'all' the related user visible parameters.
- This entails:
  - Isolating the definitions of families, types and instances.
  - Merging that data together.
  - Mapping user facing parameter names to the data columns.
- It is very important for anyone working with Exchange data to keep in mind that Revit is highly dynamic when it comes to fields/parameters.
  - End users can add new fields any time they like to a Revit model.
  - Custom fields can have duplicate Field names. Revit is able to manage this internally, and in fact does little to provide transparency to the end user that an overlapping field name state exists (anti-thesis to any reasonable data engineer).
  - Due to the possibility of overlapping names for unique fields any transformations to the data must be prepared to take these conditions into account or otherwise ignore the conditions in order to avoid exceptions in your code.

To re-iterate, taking this approach does in fact provide a high degree of visibility into any model, with the model owner able to control what is seen or shared with the consuming entity/application. The challenge is that the REST API generates enormous amounts of normalized data, and there is a good chance that you're going to need to transform all of that data to get it into some type of state that will be useful to you or your own applications needs.

GraphQL

There is however some hope on the horizon, one of the improvements to the data exchange API's that we have been discussing with Autodesk is the roll-out and availability of GraphQL API endpoint(s) to query the Data Exchanges. The GraphQL endpoints will allow the developer to perform advanced filtering as well as specify the shape of the data being returned. This will have numerous benefits, allowing developers to hone in on the specific data of interest returned in a shape they specify, thus significantly reducing both the number of API calls and the amount of data they need to extract, load and transform.

Developing

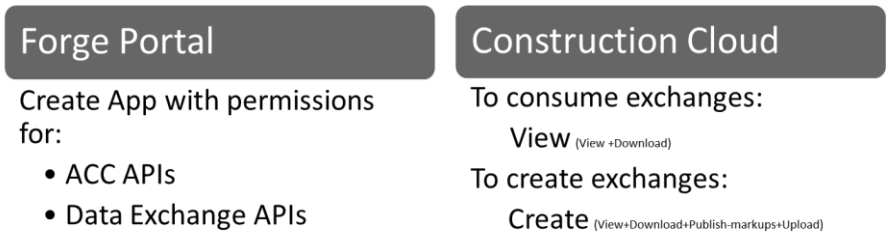
While Autodesk has provided good documentation for these new features, there is some critical information you should be aware of if you're going to embark on developing against these APIs. The most important thing to note is that both exchanges and their associated APIs are very, very new. While the data exchanges themselves have been available since April, at time of writing the published API documentation retains a very prominent disclaimer:

THIS API IS IN BETA AND THIS DOCUMENTATION IS INTENDED FOR BETA USERS ONLY

With that reminder in place, we will restrict the discussion to be somewhat generic and focused around getting up to speed so you can explore with minimal heartache.

Basics

The minimum requirement to get started is a registered Forge application and a Autodesk Construction Cloud (ACC) Project. Since this is a new set of Forge APIs, they share the same common set of protocols and authentication requirements as other forge APIs. For the uninitiated, head over to Forge developer documentation and review the Authentication Basics content to register a new forge application using the Data Exchange API and the ACC API. Once registered, ensure your application is authorized to run on your Forge tenant. On the Autodesk Construction Cloud side, make sure that the end user running the application has appropriate folder access to your test project.



Permissions Summary

Exploratory Tooling

Once the necessary permissions and app authorizations are in place, its time to start working against these APIs. At this point, there are many options available for exploration and development from cURL to full blown applications in essentially any modern language with REST support. In our case we did preliminary exploration using Postman which has great support for OAuth and browser authentication. Once we confirmed our ability to access the APIs via postman, we migrated over to working with Python via Jupyter notebooks. The notebook format provides an excellent exploratory experience while allowing us to write ad-hoc functions to do things like loop through the paginated results being returned to build a more complete picture of the contents of the data exchange.