

Session 2.4

Lab: Using Web APIs with Revit and Python

Jared Friedman, Walter P Moore

Class Description

This lab will provide an overview of how to connect your BIM content to web-based resources using REST APIs. We will utilize the open source pyRevit application and the Python Requests module in order to export and import data directly between Revit and the web. In this demo we will leverage Airtable and it's well-documented API in order to enable us to create a lightweight database that lives outside of Revit.

About the Speaker:

Jared is a licensed architect and design technologist who currently works as a Senior Technical Designer in the New York City office at Walter P Moore. Jared has taught coursework related to BIM and Computational Design at The Graduate School of Architecture, Planning and Preservation at Columbia University, and has lectured at numerous conferences and universities on the subjects. Much of his work has focused on how data and computational design processes can be applied towards topics such as automation, industrialized construction, and embodied carbon tracking in the AEC industry.



Introduction

In this lab we will be going through how to get setup to connect Revit to web-based resources using pyRevit and the Python Requests module. This guide is meant mainly to help connect the dots rather than provide a comprehensive overview of the many topics that will be covered (Web APIs, pyRevit, Python, Airtable, etc.). Fortunately, there's lots of great documentation already out there on all the topics we'll be touching on, and so you are encouraged to dive deeper into any of these things that may be new to you.

The end goal for this lab is to develop two custom tools that we will be able to use in Revit. The first tool will update some material properties in Revit that we'll be defining in Airtable. The second tool will push some changes that we'll be making in Revit back to Airtable. While this demo will focus on material properties, the same concepts could be applied to any type of data you might be looking to link between Revit and a web-based resource.

There are a handful of benefits we get from managing things like material properties outside of Revit. For starters, this will allow users to update certain information without having to open Revit. Another benefit is that we can store information that the design team may want to know, but doesn't necessarily need to show up in the documentation. Perhaps you want to store that contact information for a sales rep for a certain product - that is information that does not need to live in Revit, but would be beneficial to link to a product that exists in the model.

Before jumping into the step-by-step portion of the guide we will first overview some key concepts and steps to get setup in order to run the tools that we will be building.

Overview of Key Terms and Concepts

Key Terms

API (Application Programming Interface)

An API acts as a contract offered by an application that exposes certain functionalities of the program, allowing other applications and services to interact with it.

Web API

Web APIs are types of APIs that work with a web server or browser using HTTP protocol. These can be used for internal, private, or public consumption.

REST (Representational State Transfer)

REST (also called RESTful) APIs are an “architectural style”, most typically used with Web APIs. Simply put, it's a standard way of making API calls that many web applications have adopted.

Client

The front end or user side of the web architecture that connects to resources and services over HTTP. Your web browser is an example of a client – it interacts with APIs on different websites to get page content that is displayed in the browser.

Server

The server provides resources to be consumed by a client when a request is received through an API (without providing direct access to the content stored in the database).

General Concept of APIs

A common analogy for how these various terms work in action is placing a food order at a restaurant. Let's say we want to order a burger at a restaurant. In this case we are the **client**, the chefs preparing the food are the **server**, and the food order that we put in is the **API**. The chefs will need certain information from us such as to how we want the burger cooked (ex: medium rare with extra bacon), but we will not need to specify every detail (ex: put it on a bun, cook for four minutes on each side). This well-understood information exchange is analogous to the accepted HTTP operations that we can make.

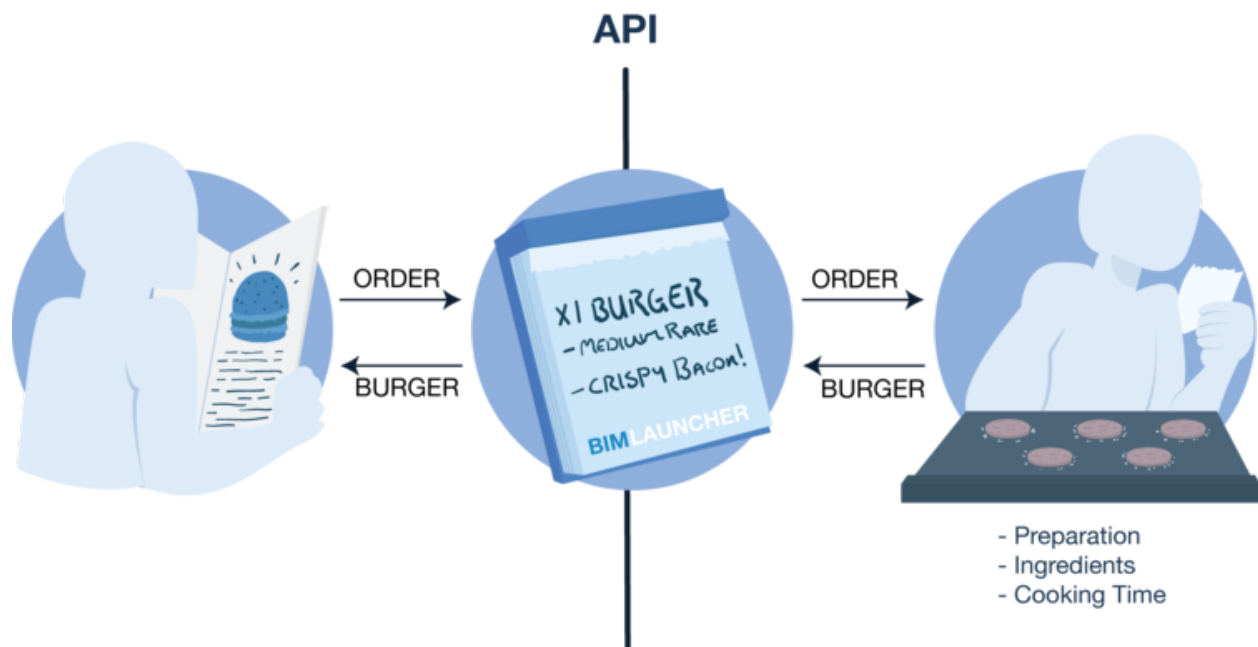


Image from <https://www.bimlauncher.com/blog/2018/1/27/forg-101-introduction-to-autodesk-forge-apis>

Primary HTTP Operations

Most of what you will need will boil down to four main CRUD (Create, Read, Update, Delete) operations when working with REST APIs. These are briefly summarized below. We will not have time to do a deep dive into how each of these operates, but there are plenty of resources out there that will explain this in greater detail.

- **GET** – Retrieve a resource
- **POST** – Create a new Resource
- **PUT** – Edit or update an existing resource
- **DELETE** – Delete a resource

The Airtable API

One of the main reasons we will use the Airtable API for this demo is that it is well documented, and relatively simple. Each Airtable base will auto-generate it's own documentation with some examples when you go to 'Help' > 'API documentation'. There's also a full guide to the documentation for developers that can be found here: <https://airtable.com/developers/web/api/introduction>

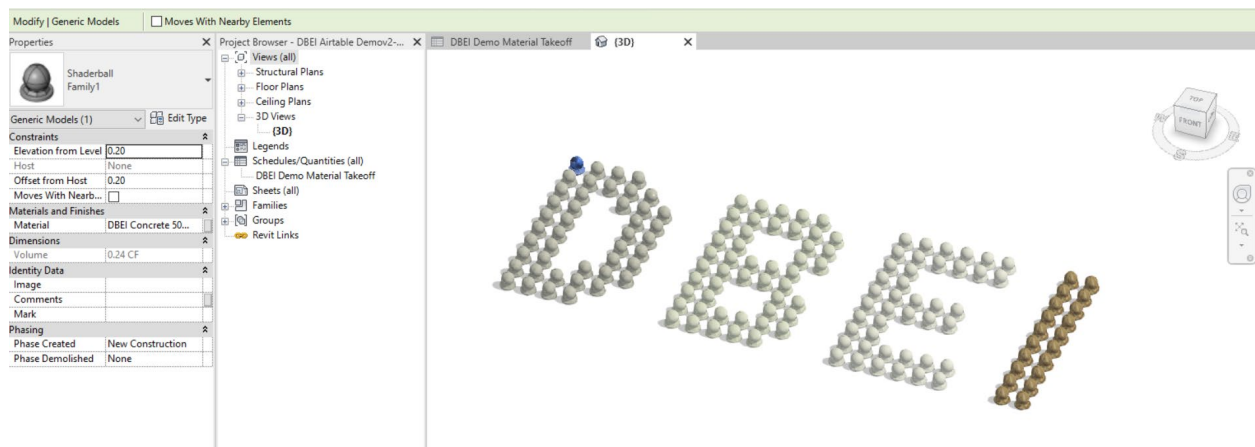
While we won't use it during this demo, there is a great Python wrapper developed by Gui Talarico and others in the community for the Airtable API called [pyAirtable](#) that can be found on github. If you plan to do further work with the Airtable API using Python, I strongly encourage you to use the pyAirtable wrapper. The wrapper will simplify a lot of the complexity in dealing with some of the intricacies of working with the API, and keep your code Pythonic. The wrapper is built for working with Python 3, but older versions will work with Python 2, which is currently the version of Python that IronPython in Revit ships with.

Setup: Installations and Links

Prior to jumping into the code we'll want to setup a few things in order for the tools we will build to work properly.

Revit File Setup

The file provided for the workshop will ready to go, but it will be good to understand what has been setup in the provided file. Generally, we'll be mapping some custom Revit Material parameters to fields in our Airtable database. Those materials are applied to some dummy elements in the model as instance parameters.

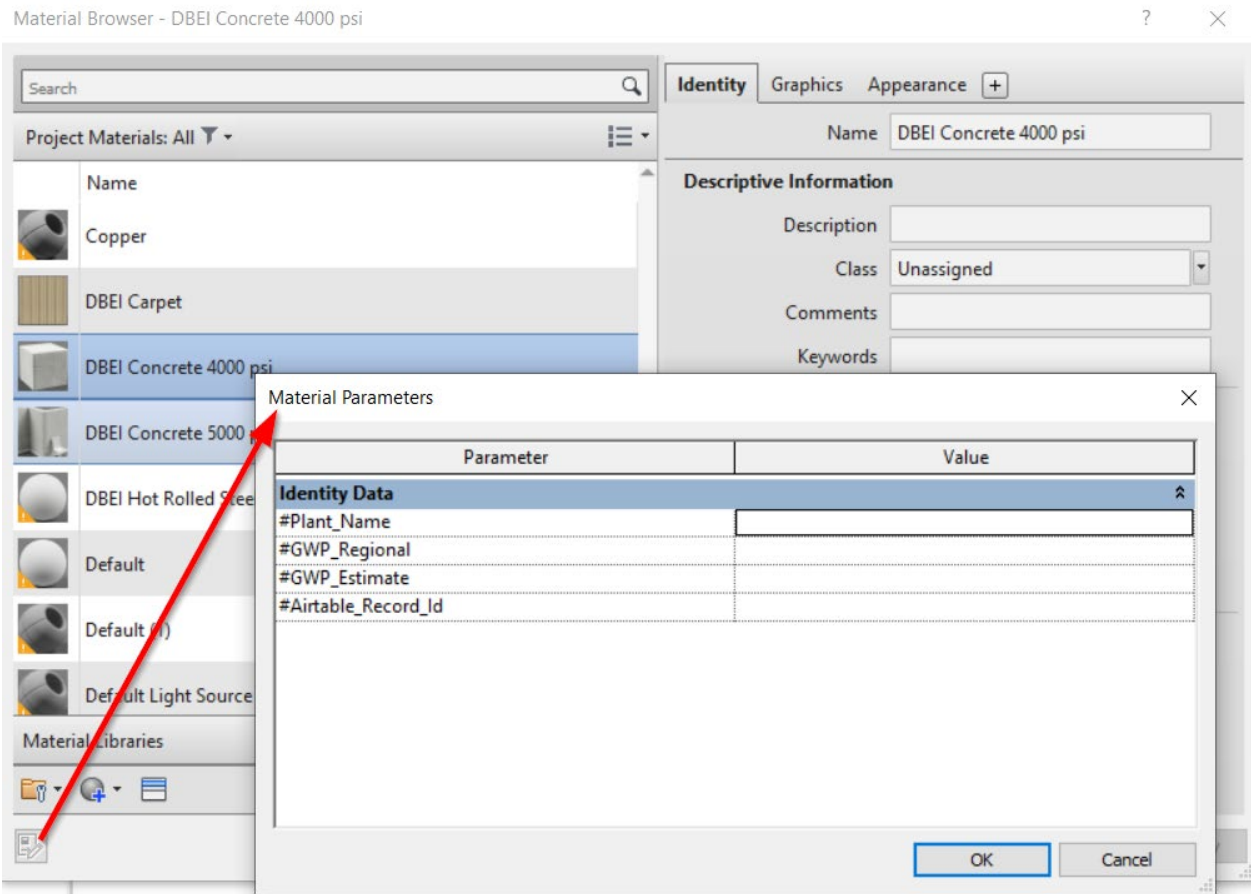


We will setup a small handful of custom material properties. A couple of these will provide numbers for GWP (Global Warming Potential) values. We won't go into depth about GWP here, but it's a helpful property to carry if you are trying to quantify embodied carbon on your project.

The following custom material properties have been setup in the project:

- `#Plant_Name` – This will store the name of the manufacturing plant
- `#GWP_Region` – This will store the our regional baseline GWP values
- `#GWP_Estimate` – This will store our estimated GWP values
- `#Airtable_Record_Id` – This will store our Airtable record ids

These parameters have been setup as Project Parameters, but they can just as easily be other parameter types. Note that we've set these up as Text parameters. We could have setup GWP (Global Warming Potential) parameters as a number as well, but keeping it as text will make future steps a bit simpler. Additionally, keeping these as text will allow us to display a variety of units since the units for GWP may vary between different material classes.



Lastly, the starter file will have a prebuilt Material Takeoff schedule in it. It will be blank for now except for the Volumes of material, which are being totaled. This schedule will make it easier for us to interact with the material property data coming in and out of Revit.

| <DBEI Demo Material Takeoff> | | | | | | |
|------------------------------|-------------|------------|--------------|--------------|----------|----------|
| A | B | C | D | E | F | G |
| MATERIAL NAME | AIRTABLE ID | PLANT NAME | GWP BASELINE | GWP ESTIMATE | VOLUME | COMMENTS |
| DBEI Carpet | | | | | 4.71 CF | |
| DBEI Concrete 4000 psi | | | | | 11.30 CF | |
| DBEI Concrete 5000 psi | | | | | 10.36 CF | |
| DBEI Hot Rolled Steel | | | | | 9.89 CF | |

pyRevit Overview and Setup

This workshop will not provide any comprehensive overview of pyRevit. There are lots of great resources out there for learning pyRevit some of which are linked below. In short, as told on their website “pyRevit (with lowercase py) is a Rapid Application Prototyping (RAD) environment for Autodesk Revit®. It helps you quickly sketch out your automation and add-on ideas, in whichever language that you are most comfortable with, inside the Revit environment and using its APIs. It also ships with an extensive set of powerful tools that showcase its capabilities as a development environment.”

pyRevit Resources:

- pyRevit Notion page: <https://pyrevitlabs.notion.site/pyrevitlabs/>
- pyRevit GitHub: <https://github.com/eirannejad/pyRevit>
- pyRevit YouTube: <https://www.youtube.com/@pyRevit>
- pyRevit API Documentation:
<https://pyrevit1.readthedocs.io/en/latest/index.html>

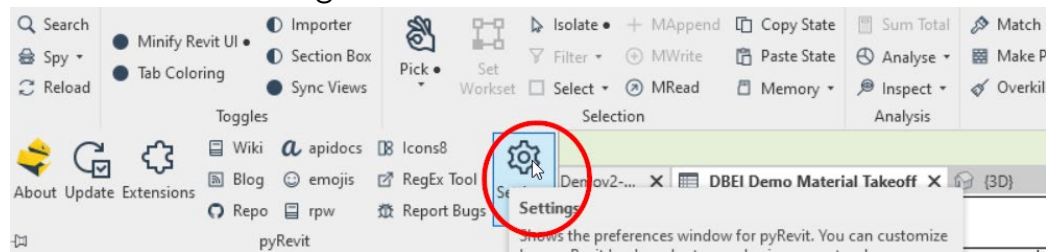
Installation

If you don't already have pyRevit installed, you can find installation instructions on the Notion home page linked above. Any of the versions put out in the last couple of years should work fine for this demo.

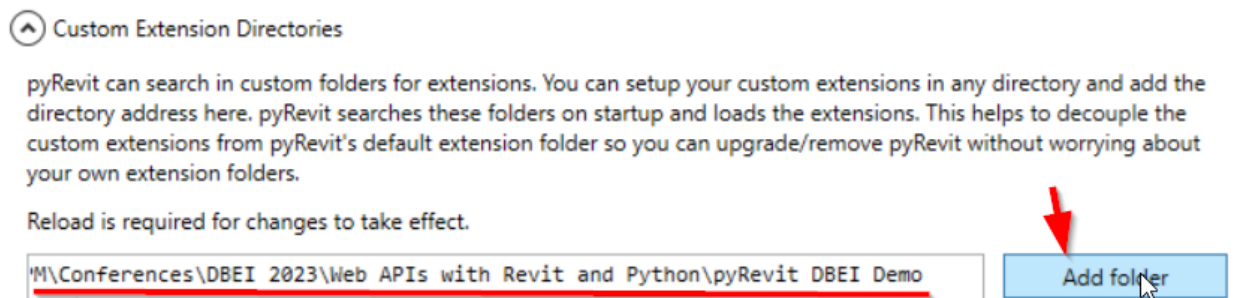
Adding a Custom Extension

1. Copy over the folder titled 'pyRevit DBEI Demo' with all of it's contents locally onto your machine. This will be where we store our custom ribbon. We will take a closer look at what's inside there in just a bit.
2. Once you have pyRevit installed, go to the pyRevit tab in Revit and hit the dropdown arrow all the way on the left next to where it says 'pyRevit'.

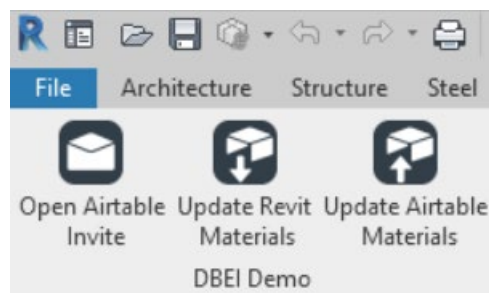
Next click on 'Settings'



3. This should open up the pyRevit Settings dialog. Scroll down to the 'Custom Extension Directories' section and click 'Add folder' and point to the folder you copied over in Step 1. (Alternatively, if you already have a custom pyRevit ribbon of your own, feel free to add the 'DBEI Demo.panel' folder to your own ribbon.)



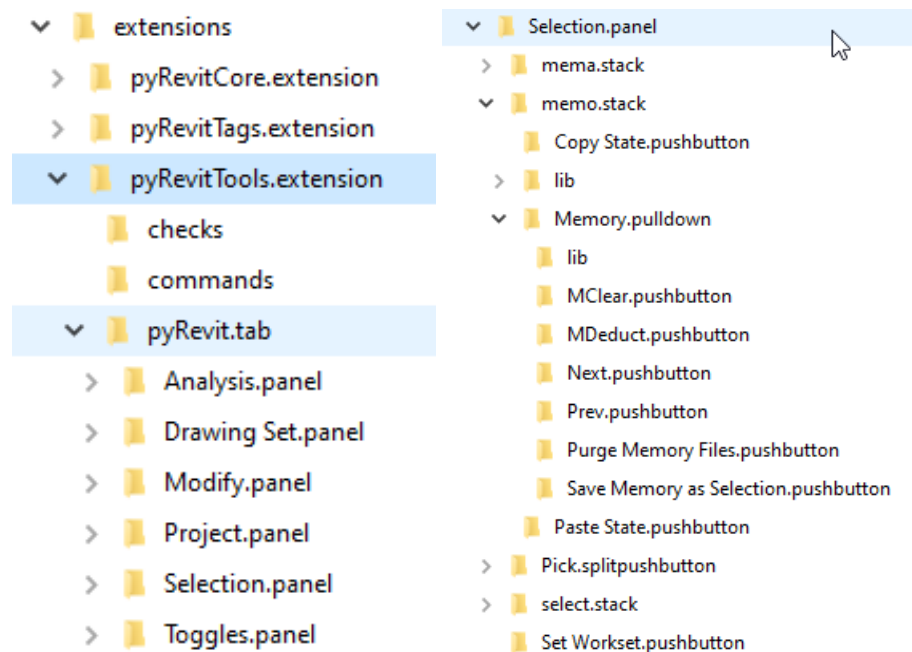
4. Click on 'Save Settings and Reload' and you should now see a ribbon in Revit called 'DBEI' with three buttons as shown below.



Brief Overview of pyRevit Folder Structure

The way in which pyRevit recognizes and organizes the ribbon is based on a specific folder structure. One of the best ways to learn the folder structure is to poke around the pyRevit ribbon folder structure. A nice tip for doing this – and generally poking around pyRevit code – is to **Alt + click** on any of the buttons in the pyRevit ribbon, which will open up the folder where it lives locally on your machine. The following image shows what the pyRevit folder structure looks like.

Generally the folder hierarchy goes XXX.extension > XXX.tab > XXX.panel. Within the panels you can have dropdowns, stacks and pushbuttons.



Airtable Access and Setup

Airtable is an easy-to-use online platform used for setting up lightweight relational databases. Since it's a flexible tool with a well-documented API, it's a nice tool for the purposes of demonstrating how to work with web APIs.

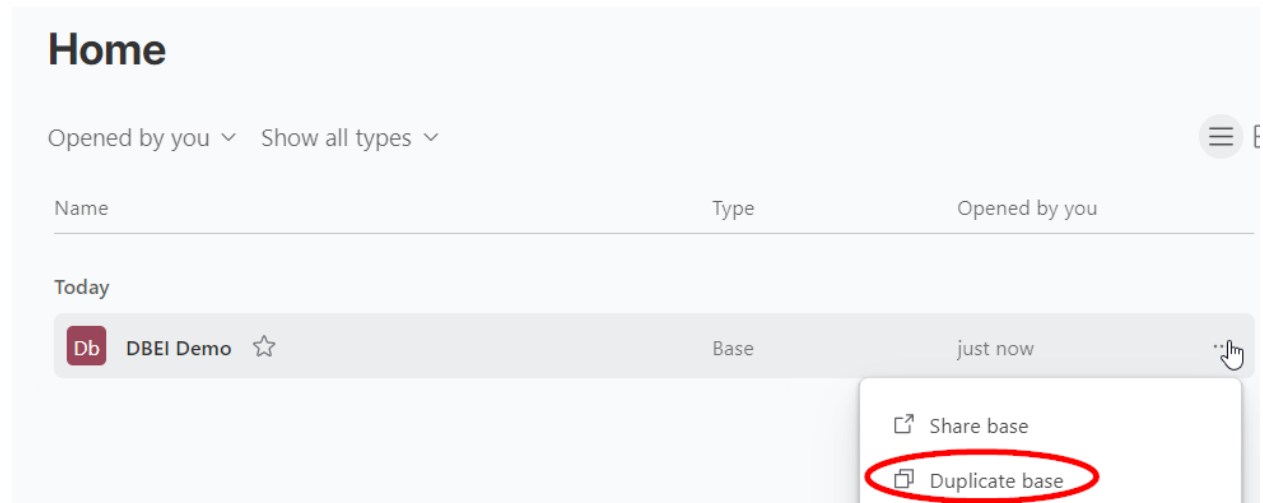
An Airtable base has been setup for this demo that each person following along should create their own copy of. You can access an invite to the base by clicking on the 'Open Airtable Invite' in your DBEI ribbon that was setup in the previous step. (Link is here as well in case you just want to poke around the Airtable base that's been setup: <https://airtable.com/shrJTR5wQHATMQFHs>)

Duplicate the Base

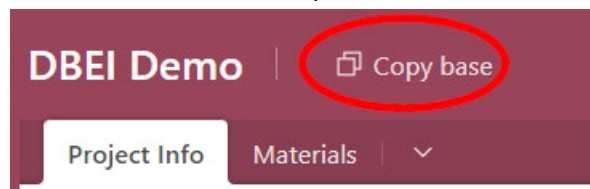
Duplicating the base will give you your own base to begin using throughout this demo. Follow the steps below to duplicate the base.

Go to your Airtable home page where you should see a list of all your Bases and Workspaces. Find the DBEI Demo base and click on the three dots to the right, and then 'Duplicate Base'. This will create a copy of the base. You can rename

it whatever you'd like.



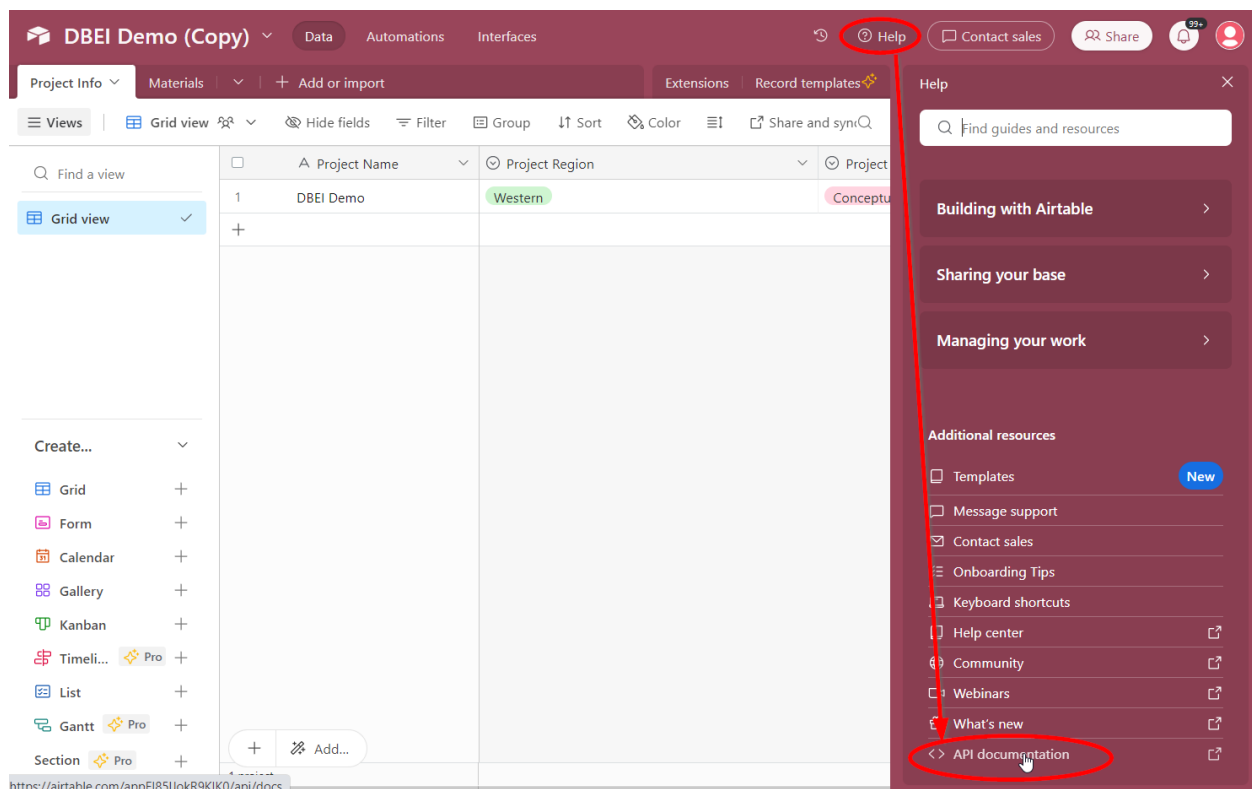
Alternatively, if using the link from above instead of the Airtable invite from the button in Revit, you would click on 'Copy Base':



Find the Base Id

Each Airtable base contains a unique id that shows up in the URL of the base, and is needed for connecting via the API. You can find your base id by going to 'Help' at the upper right, and then 'API documentation' at the lower right. This will launch the very helpful documentation page where you can find the base id. It should look something like this: `appEI85lJokRABCDE`

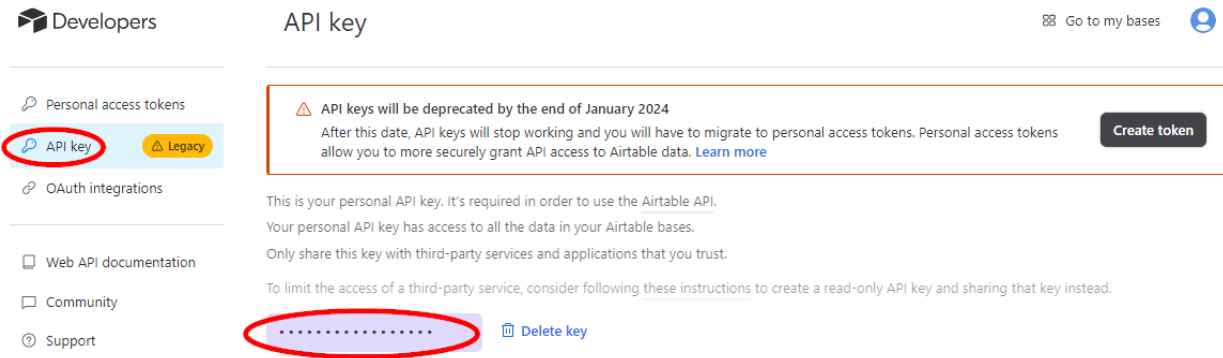
Copy and paste this id somewhere for a future step.



Find Your Personal API Key

Airtable is in the process of moving over to personal access tokens instead of a single user API Key. For now, we will work with the API Key approach since it's a little simpler. The switch to personal access tokens won't significantly change the instructions here in the future, but be aware of this change if you plan to use the Airtable API in the future.

To find your API Key, go to the Developer Hub by clicking the user icon in the upper right and hitting 'Developer hub'. Next, go to 'API Key' on the left-hand side of the page, and then click inside the hidden token to see and copy it. Copy and paste this id somewhere for a future step.



Developers

API key

Go to my bases

Personal access tokens

API key Legacy

OAuth integrations

Web API documentation

Community

Support

API keys will be deprecated by the end of January 2024

After this date, API keys will stop working and you will have to migrate to personal access tokens. Personal access tokens allow you to more securely grant API access to Airtable data. [Learn more](#)

Create token

This is your personal API key. It's required in order to use the Airtable API.

Your personal API key has access to all the data in your Airtable bases.

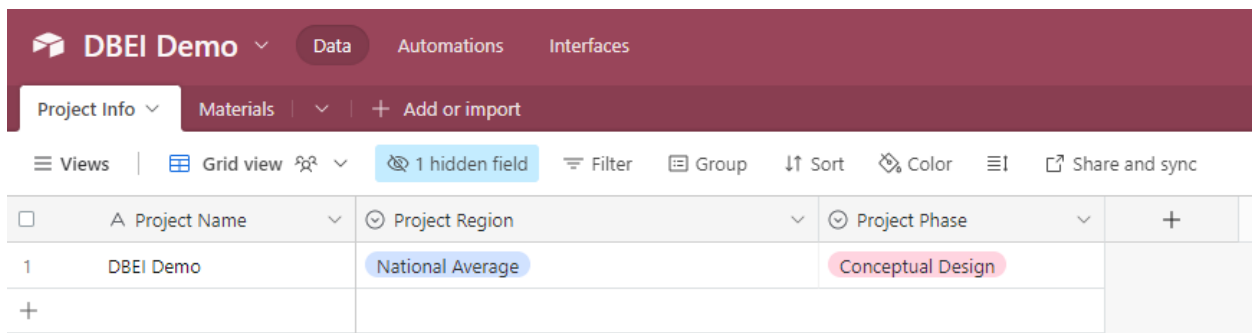
Only share this key with third-party services and applications that you trust.

To limit the access of a third-party service, consider following [these instructions](#) to create a read-only API key and sharing that key instead.

..... Delete key

Overview of the DBEI Demo Base

The base that has been setup for this demo consists of two tables: 'Project Info' and 'Materials'. The 'Project Info' table will store our general Project Region and our Project Phase. These are both Single Selection fields that will allow us to switch between regions and project phases. Based on the values selected for these fields, the values of some of the fields in the Materials table will be altered. This is to highlight the relational aspect of Airtable bases.



| | Project Name | Project Region | Project Phase | |
|---|--------------|------------------|-------------------|--|
| 1 | DBEI Demo | National Average | Conceptual Design | |
| + | | | | |

The second table, titled 'Materials' is where we will store our material "records". Each record represents a unique material, and we've defined a number of properties for each material (*Note that the GWP values plugged in for these materials are not actual standard numbers and should not be used on real projects). A few of the fields have been prefixed with a '#' symbol. This is not required, but is just being used here to tell us which properties will map to properties that exist in Revit.

| DBEI Demo | | | | | | | |
|--|------------------------|-------------|--------------------|-----------|------------------|-------------------|----------------|
| Data Automations Interfaces | | | | | | | |
| Project Info Materials Add or import | | | | | | | |
| Views Grid view Hide fields Filter Group Sort Color Share and sync | | | | | | | |
| | A Material Name | A GWP Units | A GWP CLF Baseline | lookup | Project Region | Project Phase | A GWP National |
| 1 | DBEI Concrete 5000 psi | kgCO2e/yd3 | 443 kgCO2e/yd3 | DBEI Demo | National Average | Conceptual Design | 279 |
| 2 | DBEI Concrete 4000 psi | kgCO2e/yd3 | 359 kgCO2e/yd3 | DBEI Demo | National Average | Conceptual Design | 236 |
| 3 | DBEI Hot Rolled Steel | kgCO2e/lb | 0.771 kgCO2e/lb | DBEI Demo | National Average | Conceptual Design | 0.65 |
| 4 | DBEI Carpet | kgCO2e/ft2 | 1.86 kgCO2e/ft2 | DBEI Demo | National Average | Conceptual Design | 1.50 |
| + | | | | | | | |

One of the many powerful features in Airtable is the ability to link to records in other tables and lookup data belonging to that record. In our case we've linked our materials to the 'DBEI Demo' project record, and are doing lookups into the 'Project Region' and 'Project Phase' fields of those project records.

| lookup | Project Region | Project Phase |
|------------------------|------------------|-------------------|
| Link to another record | National Average | Conceptual Design |
| DBEI Demo | National Average | Conceptual Design |

Another powerful feature of Airtable is to put in complex formulas for fields. We use the 'Formula' field type for doing this. In the example below we will switch between the different regional GWP numbers we've defined based on the region that the project has been set to. We will not do a deep dive into what's possible with formulas, but if you're curious to learn more about them you can find further documentation here: <https://support.airtable.com/v1/docs/formulas>

| | |
|-----------------------|------------------------|
| Assigned Regional GWP | #GWP Regional Baseline |
|-----------------------|------------------------|

Assigned Regional GWP

Formula

Compute values based on fields. [Learn more](#)

Formula

```
SWITCH({Project Region}, 'National Average', {GWP National Average}, 'Eastern', {GWP East Baseline}, 'Central', {GWP Central Baseline}, 'Western', {GWP West Baseline})
```

Scripting the Tools (the fun part!)

The following section will highlight some of the key things happening in the code in order to work with Web APIs with Python and Revit. This will not be a comprehensive overview of Python and will not be going through line-by-line. There are plenty of other great resources out there if you are just getting started with Python and pyRevit that I encourage you to check out.

We will be building two tools, one which will pull some data from Airtable to Revit, and one that will update Airtable records from Revit. Since both of these tools will be accessing the same Airtable base, we'll keep some of the references in a separate file, and we'll import the variables into each tool as required.

Airtable References (my_airtable.py)

This portion of the tools can be found inside the 'dbei.extension\DBEI.tab\DBEI Demo.panel\Lib' folder.

This short Python script contains info that is required in both of our tools. Placing it once here will save us some repetition between scripts. Here is where we will paste in our API Key and Base Ids that we copied over in previous steps.

The final URL we want should include the Airtable base URL, the version (v0), base Id, and table name. The table name is just the name of our table in Airtable. I'd recommend avoiding spaces in the table name for purposes of this demonstration.

The headers will include a Bearer schema for authentication. This will change in future versions of Airtable as previously mentioned, but it is the schema that we'll use for this demo. The Bearer schema allows the client to access resources that would otherwise require credentials from the user.

```

import posixpath

BASE_ID = "app7GoC6pa9EAbcdE" # Replace this with your own base id
AIRTABLE_API_KEY = "keyW2ab1I9mhAbcdE" # This is a fake key
TABLE_NAME = "Materials"

VERSION = "v0"
API_BASE_URL = "https://api.airtable.com/"

url = posixpath.join(API_BASE_URL, VERSION, BASE_ID, TABLE_NAME)

headers = {"Authorization": "Bearer {}".format(AIRTABLE_API_KEY)}

```

Update Revit Materials (update_rev_mat_script.py)

This portion of the tools can be found inside the 'dbei.extension\DBEI.tab\DBEI Demo.panel\Update Revit Materials.pushbutton' folder.

This Python script will comprise of three general steps:

1. Collect all the records from the Airtable 'Materials' table
2. Ask the user what materials they would like to update
3. Collect and update the desired materials in Revit.

Prior to these steps we will do our general imports, our imports from the my_airtable.py file, and setup a Python dictionary that maps our Airtable field names (the names of the columns in Airtable) to Revit parameter names. Note that we will need to set `__fullframeengine__` to **True** in order to allow pyRevit to work with the Python 'requests' module.


```

from my_airtable import url, headers

__fullframeengine__ = True #must be set in order for requests to work

AT_TO_REVIT_MAPPING = {
    "#GWP Regional Baseline": "#GWP_Regional",
    "#GWP Conservative Estimate": "#GWP_Estimate",
    "#Material Plant": "#Plant_Name",
    "Airtable Id" : "#Airtable_Record_Id"
}

```

Step1: Collecting Airtable Records

This is where the fun part happens! Here is where we will use the 'requests' module in order to make a GET request from Airtable. For this demo we will keep things simple and just return all the records. Often when making a GET request you will also pass additional parameters (in a keyword argument called 'params') in order to filter what is returned.

The response we will get from Airtable is itself an object that we can extract a json from. Json files in Python can be used very similarly to dictionaries. The full json response will contain some other metadata about the base that is not needed for our purposes, so we have specified to just get the list of records ("records" is the key that the Airtable json will return with the list of records as the value).

```

def get_request_requests(request_url):
    """
    Make a get request using Python requests module
    """
    headers = {"Authorization": "Bearer {}".format(AIRTABLE_API_KEY)}
    response = requests.get(request_url, headers=headers)
    json_response = response.json()

    return json_response

```

```
airtable_response = get_request_requests(url)
all_mat_records = airtable_response["records"]

mat_dict = process_records(all_mat_records)
```

The `process_records` function will take the list of Airtable records and strip them down to the relevant info in the format we desire. Below is what a single record from our Airtable base might look like.

```
{
  "id": "rec2iGnr1dHEcKiz7",
  "fields": {
    "#Material Plant": "Sizzle Steelworks",
    "GWP National Average": "0.65",
    "Assigned Regional GWP": "0.60",
    "GWP Units": "kgCO2e/lb",
    "Material Name": "DBEI Hot Rolled Steel",
    "GWP Specified": 0.57999999999999996,
    "Project Region": [
      "Western"
    ],
    "#GWP Regional Baseline": "0.60 kgCO2e/lb",
    "#GWP Conservative Estimate": "0.73 kgCO2e/lb",
    "GWP East Baseline": "0.65",
    "GWP Central Baseline": "0.75",
    "Lookup": [
      "rec0CYXo29GoJuY7R"
    ],
    "GWP West Baseline": "0.60",
    "GWP Specified Plus Uncertainty": 0.72499999999999998,
    "Project Phase": [
      "Conceptual Design"
    ],
    "GWP CLF Baseline": "0.771 kgCO2e/lb"
  },
  "createdTime": "2023-05-21T04: 43: 19.000Z"
}
```

The returned dictionary should look something like what we see below. This has just the properties that we will attempt to update in Revit. Note that we've stored the Airtable Record Id as a property. This may feel a bit odd now, but this will help us in the next tool we build when we want to update the records.

```
{
  "DBEI Carpet": {
    "Airtable Id": "recVEw0v20PY1arWP",
    "#GWP Conservative Estimate": "1.85 kgCO2e/ft2",
    "#GWP Regional Baseline": "1.45 kgCO2e/ft2",
    "#Material Plant": "Snuggle Strands"
  },
  "DBEI Concrete 5000 psi": {
    "Airtable Id": "recUJVyjsxCveh4FiP",
    "#GWP Conservative Estimate": "331.88 kgCO2e/yd3",
    "#GWP Regional Baseline": "290 kgCO2e/yd3",
    "#Material Plant": "Mix-a-lot Concrete"
  }
}
```

Step2: Ask User What to Update

We may or may not want to update all of the materials, so here is where we'll ask the user which ones they want to update. pyRevit has a number of handy built-in forms inside the 'forms' module that we imported. Here we use the [SelectFromList](#) class, and pass it the keys from our materials dictionary that we ended with in the previous step. This will return a list of just the selected materials that we will retrieve in the last step.

```
mats_to_update = forms.SelectFromList.show(
    sorted(mat_dict.keys()),
    multiselect=True,
    button_name="Update Materials",
    title="Select Materials to Update"
)
```

Step3: Collect and Update Revit Materials

In this last step we'll collect the materials that the user checked. Inside of a Transaction we'll loop through the materials, and for every property that we mapped, we will update it in Revit. Notice that we use the 'db' module from rpw for both the Collector and Transaction. This can be done without the 'db' module, but using it makes the code a bit more Pythonic and easier to read. The rpw library ships with pyRevit by default.

```
mat_collector = db.Collector(of_class="Material",
                             where=Lambda x: x.Name in mats_to_update)

with db.Transaction("Updating Revit Materials"):
    for mat in mat_collector:
        property_dict = mat_dict[mat.Name]
        for key, value in property_dict.items():
            rev_param_name = AT_TO_REVIT_MAPPING[key]
            rev_param = mat.LookupParameter(rev_param_name)
            rev_param.Set(value)
```

Update Airtable Materials (update_air_mat_script.py)

This portion of the tools can be found inside the 'dbei.extension\DBEI.tab\DBEI Demo.panel\Update Airtable.pushbutton' folder.

This Python script will be very similar to the previous tool, but in reverse.

1. Collect materials in Revit that have Airtable Ids specified
2. Ask the user what materials they would like to update in Airtable
3. Update the matching Airtable record

Similar to what we did at the beginning of the previous tool, we will start by doing our general imports and our imports from the my_airtable.py file. This time – for simplicity – we will only be updating the OOTB 'Comments' property, so we

do not need to make a mapping dictionary for this tool. Remember to set `__fullframeengine__` to **True** again in order to allow pyRevit to work with the Python 'requests' module.

Step1: Collect Revit Materials with Record Ids

We only want to attempt to update a material if it has a value for the “#Airtable_Record_Id” property. Again, we'll use the Collector from the `rpw.db` module to collect these materials. Note that we've made a function to check if the material contains a valid record id. We'll also create a new dictionary with the material name as the key. This dictionary will make the next steps easier.

```
mat_collector = db.Collector(of_class="Material",
                             where=lambda x: check_for_record_id(x))

mat_dict = {m.Name : m for m in mat_collector}
```

Step2: Ask User What to Update

Same as we did in the previous tool, but this time the list of materials is coming from Revit as opposed to Airtable.

Step3: Update records in Airtable

In this step we'll use the PATCH request to update specific fields in Airtable. The PATCH request differs from PUT in that we can target just specific fields to update rather than overwriting the entire record.

Since the only field we're updating is the “#Comments” field in Airtable, we first make a simple dictionary that maps the Airtable Record id to the Comment coming from Revit. The #Comment field is just a simple text field in Airtable so we won't run into any issues, but you should be aware of the data type of the field you are trying to write to. More complex field types in Airtable such as formulas could become tricky to write to.

We've defined a function to update the records as shown below. Note that since we only are updating a small number of records in this demo, we are avoiding having to deal with things like pagination where we can only send a

limited number of records at a time. A more robust approach to this would batch the updates into groups until you run out of records. The shortcut approach taken here is just done to highlight the ability to update records using requests.

```
def patch_request_requests(request_url, data):  
    """  
    Make a patch requests using Python requests module  
    """  
    json_data = json.dumps(data)  
    headers = {  
        "Authorization": "Bearer {}".format(AIRTABLE_API_KEY),  
        "Content-Type": "application/json"  
    }  
  
    response = requests.patch(request_url, data=json_data, headers=headers)  
    return response
```

```
def update_records(url, headers, record_dict):  
    """  
    Places a PATCH request in order to update existing records in Airtable  
    """  
    for id, data in record_dict.items():  
        record_url = posixpath.join(url, id)  
        fields_dict = {"fields" : {"#Comments": data}}  
  
        patch_request_requests(record_url, fields_dict)
```

Running the Tools (the easy part!)

Now that we're all good to go we can run the tools. You will need to run the 'Update Revit Materials' tool prior to running the 'Update Airtable Materials' tool. This is because the 'Update Airtable Materials' tool will be looking for the Airtable Record ids that should have been injected by the first tool. It will be helpful to have the 'DBEI Demo Material Takeoff' schedule up as we're testing the tool to see the changes occurring.

To test if things are working as expected, run the 'Update Revit Materials' tool. We should see the schedule populate as seen below (values may vary depending on the Project Info you setup).

| <DBEI Demo Material Takeoff> | | | | | | |
|------------------------------|-------------------|--------------------|-----------------|-------------------|----------|----------|
| A | B | C | D | E | F | G |
| MATERIAL NAME | AIRTABLE ID | PLANT NAME | GWP BASELINE | GWP ESTIMATE | VOLUME | COMMENTS |
| DBEI Carpet | recVEw0v20PY1arWP | Snuggle Strands | 1.50 kgCO2e/ft2 | 1.85 kgCO2e/ft2 | 4.71 CF | |
| DBEI Concrete 4000 psi | recZb6wRgPGKXpXLT | Mix-a-lot Concrete | 236 kgCO2e/yd3 | 307.63 kgCO2e/yd3 | 11.30 CF | |
| DBEI Concrete 5000 psi | recUJVyxCveh4FIP | Mix-a-lot Concrete | 279 kgCO2e/yd3 | 331.88 kgCO2e/yd3 | 10.36 CF | |
| DBEI Hot Rolled Steel | rec2IGnr1dHEcKiz7 | Sizzle Steelworks | 0.65 kgCO2e/lb | 0.73 kgCO2e/lb | 9.89 CF | |

Now let's demonstrate the power of Airtable's relational databases. Go in and change the Project Region and/or the Project Phase in the 'Project Info' table. Now run the 'Update Revit Materials' tool once more, and you should see all of the GWP numbers update. This demonstrates that a simple change of one or two pieces of data in Airtable can quickly allow us to update many more properties in Revit at the click of a button.

Now, let's test the ability to push data back into Airtable. Fill in some values for the Comments in all or some of the materials directly in the Revit schedule. Open the Materials table in Airtable and scroll over to the #Comments field to watch it update in real time. Now, click the 'Update Airtable Materials' button and check all the materials we added comments to. You should see the comments populate in Airtable. Yay!

| <DBEI Demo Material Takeoff> | | | | | | |
|------------------------------|-------------------|--------------------|-----------------|-------------------|----------|----------|
| A | B | C | D | E | F | G |
| MATERIAL NAME | AIRTABLE ID | PLANT NAME | GWP BASELINE | GWP ESTIMATE | VOLUME | COMMENTS |
| DBEI Carpet | recVEw0v20PY1arWP | Snuggle Strands | 1.50 kgCO2e/ft2 | 1.7 kgCO2e/ft2 | 4.71 CF | This |
| DBEI Concrete 4000 psi | recZb6wRgPGKXpXLT | Mix-a-lot Concrete | 240 kgCO2e/yd3 | 283.02 kgCO2e/yd3 | 11.30 CF | is |
| DBEI Concrete 5000 psi | recUJVyxCveh4FIP | Mix-a-lot Concrete | 290 kgCO2e/yd3 | 305.33 kgCO2e/yd3 | 10.36 CF | so |
| DBEI Hot Rolled Steel | rec2IGnr1dHEcKiz7 | Sizzle Steelworks | 0.65 kgCO2e/lb | 0.67 kgCO2e/lb | 9.89 CF | fun |

Conclusion

We ran through quite a lot, but hopefully this provides an overview of the power of connecting Revit to web-based databases. There's plenty of improvements we can make to the code in order to make it more robust. Our tools currently assume a lot of things such as parameters existing in the model, a fairly small number of records we're working with, and that the tools are run in a specific order. These are things we'd want to better address if we were to deploy these tools within a firm. These types of things can be handled with further checks in the code and by leveraging other resources such as pyAirtable to provide more comprehensive handling of the Airtable API. Many of the things covered throughout this tutorial, such as working with the Python requests module, are not unique to working within Revit and can be leveraged in a variety of platforms. Hopefully you've found this lab helpful, and happy coding!