Smart City Fans Hackathon 22 June 2017

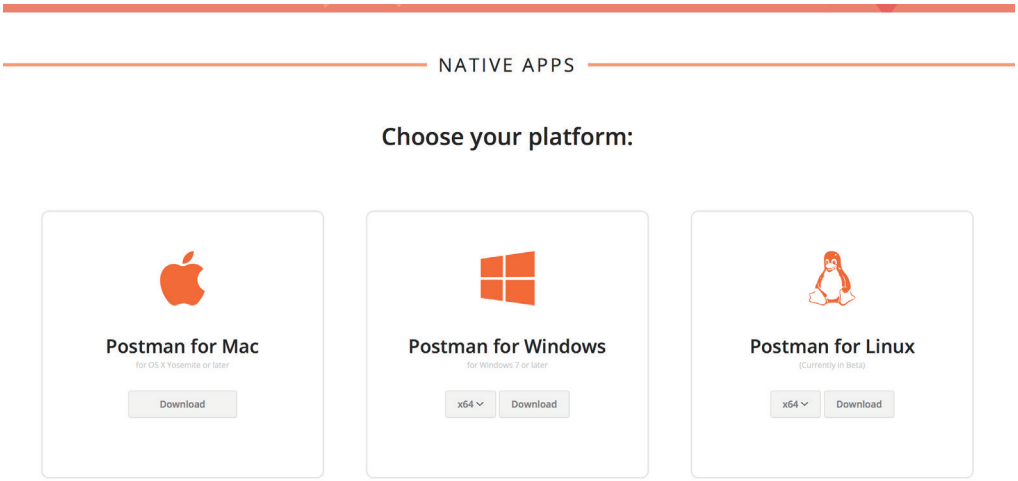
Step 1: install Postman

What is Postman?

“A powerful GUI platform to make your API development faster & easier, from building API requests through testing, documentation and sharing. Postman doesn’t require learning a new language, complicated UI, or new workflows. Developers can start using Postman immediately to make API development faster & easier.”

https://www.getpostman.com/docs/postman/launching_postman/installation_and_updates

Postman recommends using Postman native apps.
I downloaded the free Postman app on a desktop macOS Sierra.



Postman native apps are built on Electron.

What is Electron?

Electron is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. It takes care of the hard parts so you can focus on the core of your application.

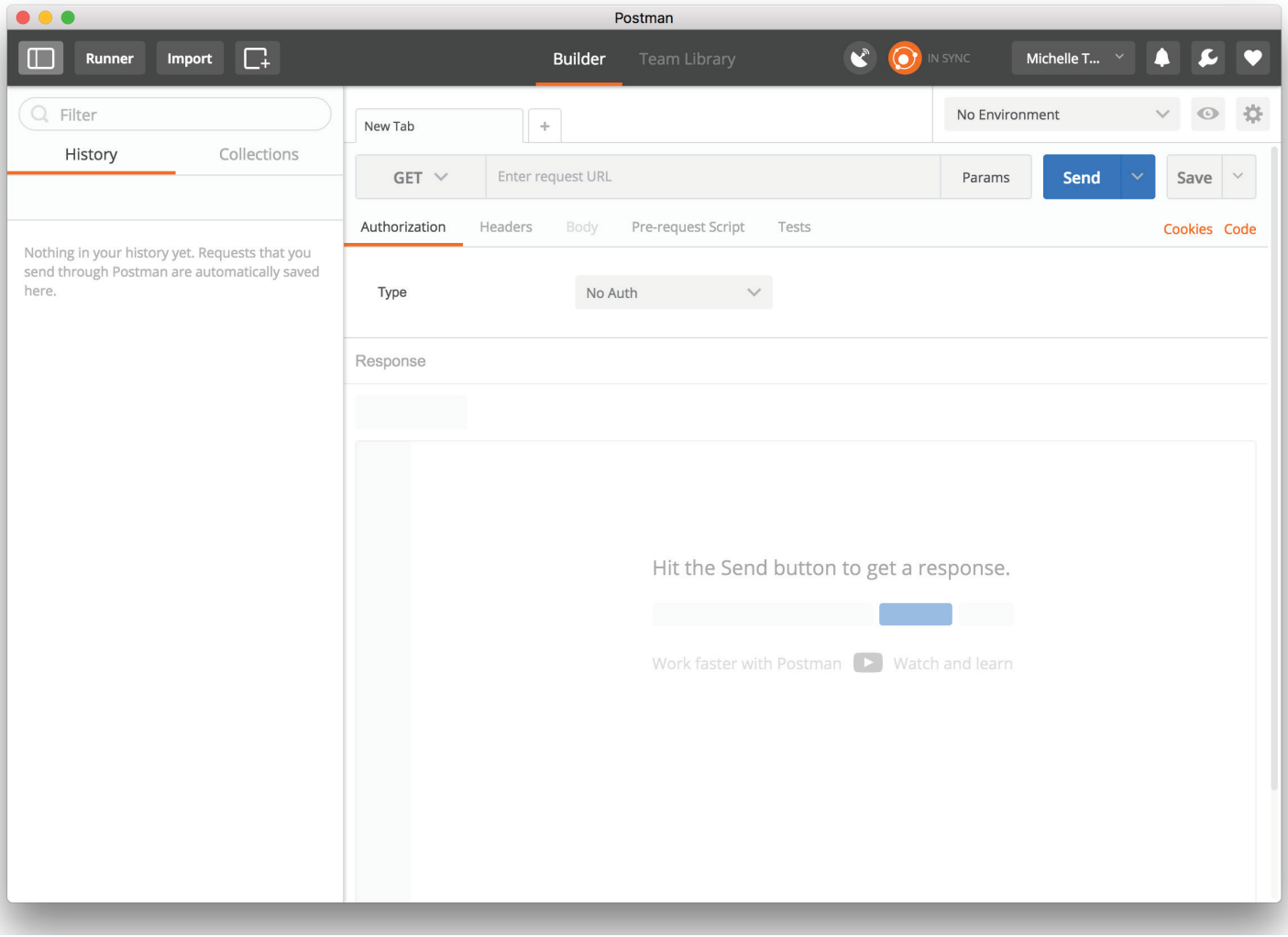
https://electron.atom.io

Reading abotu Electron didn't immediately reveal anything that wasn't already available in Postman.

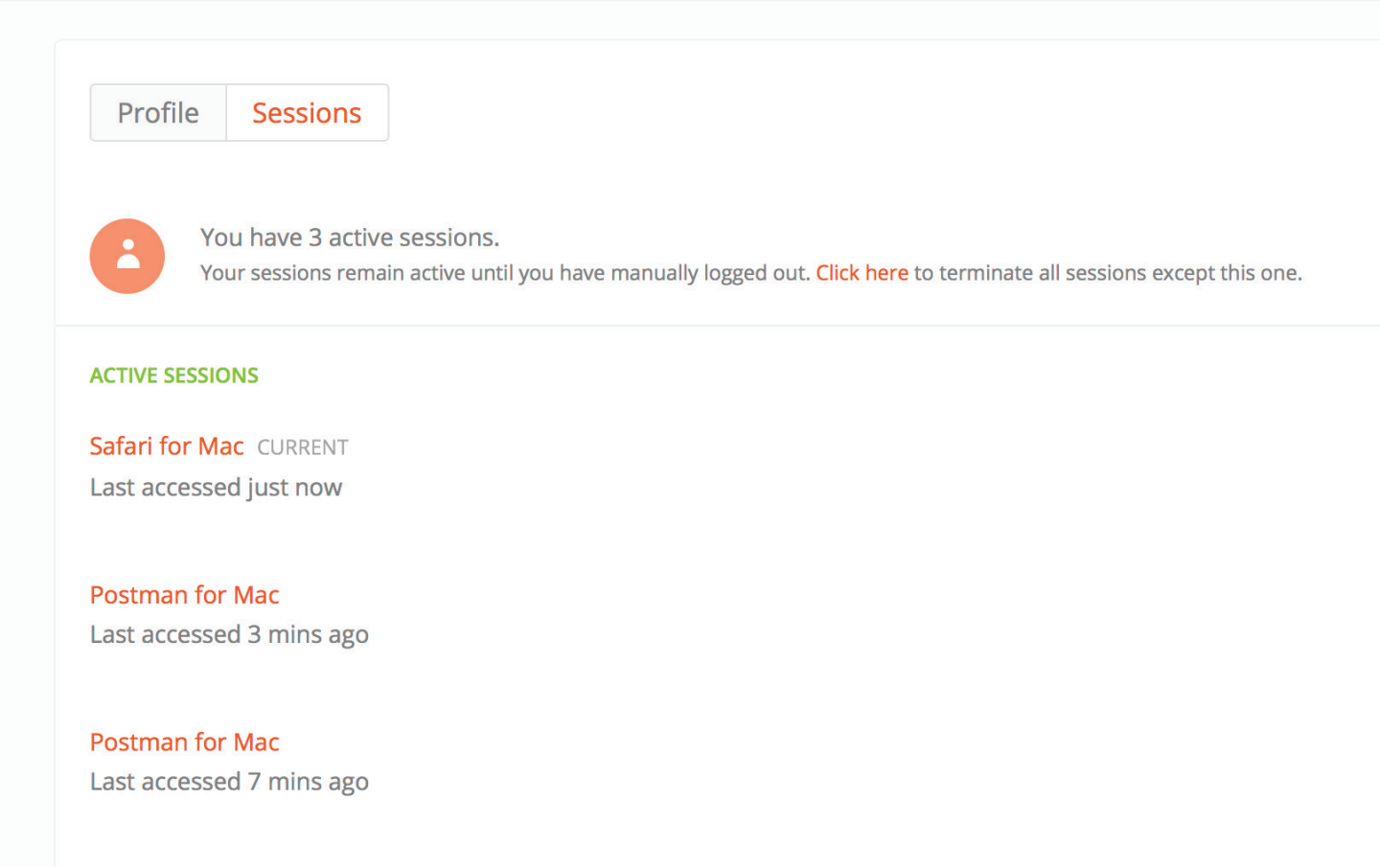
When you install Postman, you "sign in". There's an option to "sign in with Google". I used my Google credentials.

Next is "Set up account" where you pick a unique username. I picked Abraxas3d and hit enter. Dialogue box with title of "Set up account" went white and stayed white for several minutes. Canceled box. Repeated Google sign-in. Re-entered credentials. It took me straight to my account on the dashboard of the Postman app instead of "Set up account".

Here's a clean dashboard view of Postman. This is what the demo last night was using to get data from the street light "box".



The Postman account page (launched from your name in upper right of Postman app) shows your Profile and your Sessions. You can terminate Sessions from this page in case they get out of hand.



Next step: download and install Postman collections and install the Current by GEspecified Postman environment.

Both the collections and environment are given in the links. From the FAQ:

"You need to install collections into Postman for everything to work properly. Collections allow you to organize your calls (GET, POST, etc.) from a single location. Download the json files (Collections and Environment). OpenPostman, clickFile /Importto open the Importer. You can select thefilesfrom where you have stored them, or drag and drop the Collection and Environment files into the Importer. Postman will automatically load the files."

These were a direct download from the Dropbox links provided. The import button is upper left. The collection imported without an error. The environment imported and declared success but also threw an error alert.

The collections showed up on the left-hand pane when

“Collections” tab was clicked. The environment showed up under the drop-down box on the upper right. It says No Environment until you select an environment.

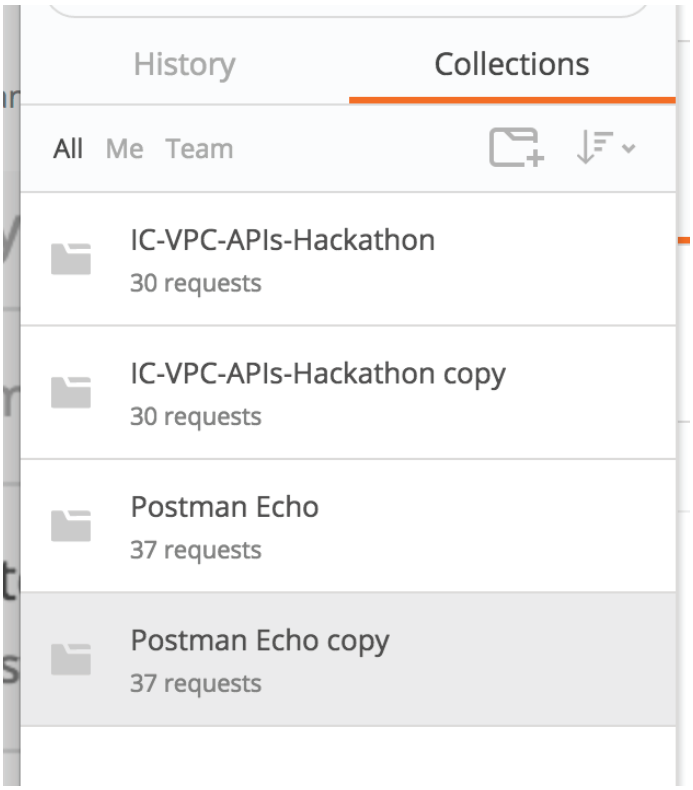
With the collections showing up and the environment set, confidence was high that the Postman app was ready for development.

What are Collections?

A Postman Collection lets you group individual requests together. These requests can be further organized into folders.

The point of a Collection is to organize related requests, document the API, easily attach test scripts, and have production scripting that passes data between API requests.

When you build a new API request in the builder, you can save it. You can also save API requests from the History tab by using Save to Collection. You can also duplicate an entire collection. Not a bad idea to make a duplicate of the newly-installed Collection just in case the working Collection gets bolluxed up later.



What is the Environment?

You will need to setup your environment so it can utilize variables in the API calls.

Without the Collections and the Environment, Postman will not work right for the Hackathon.

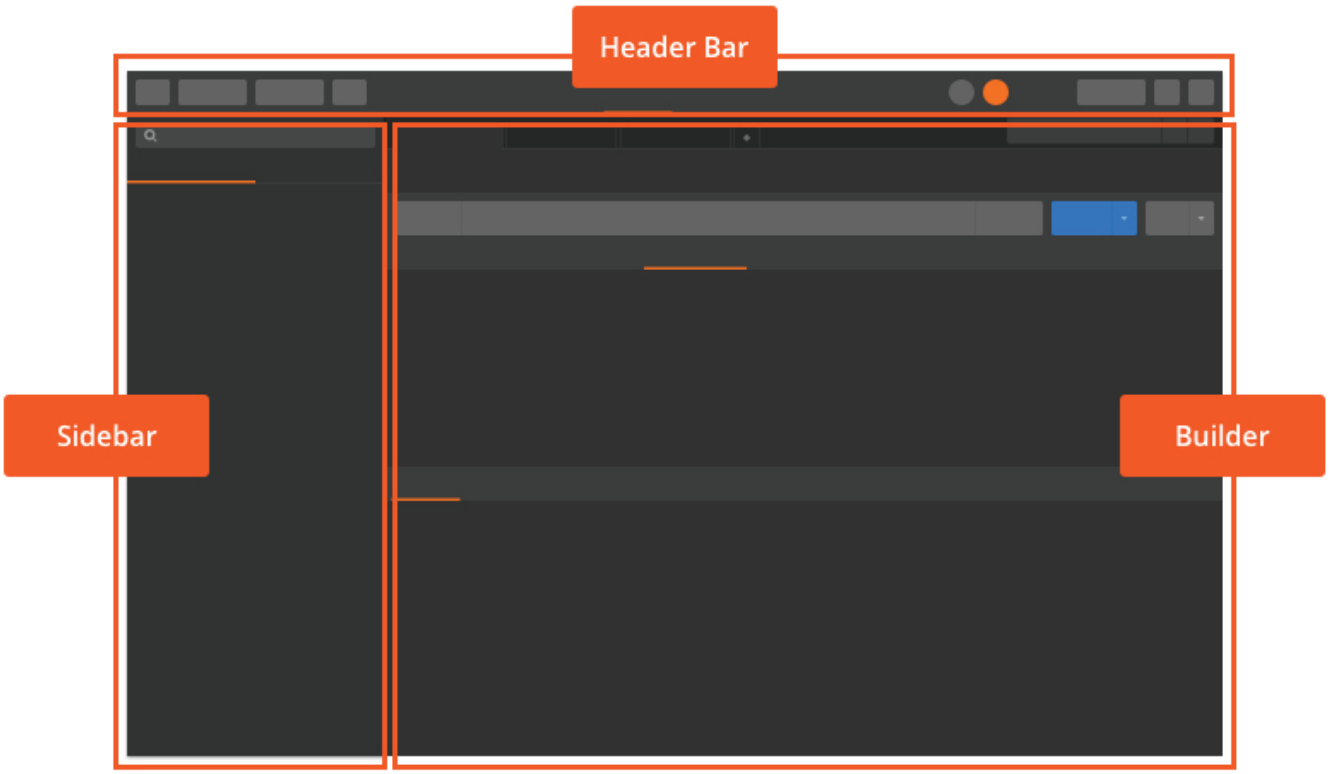
Each environment is a set of key-value pairs, with the key as the variable name. These can be edited using the data editor.

Environment and global variables will always be stored as strings. If you’re storing objects/arrays, be sure to JSON.

stringify() them before storing, and JSON.parse() them while retrieving.

Environments are managed from a drop-down box in the upper right of the dashboard.

Click the Quick Look (the eyeball) icon in the upper right corner of the Postman app to display the environment and global variables. Clicking on the Edit link will open a modal for editing keys and values.



Above is the layout of the dashboard. If you have ever seen an IDE before, then this is a familiar design pattern. Sidebar pane has History and Collections. Builder pane is where editing happens. Header Bar hears the lamentations of your enemies as they fall before your mighty coding powers. In the Header are some useful things. The Runner button opens the Collection Runner. Import does file imports. Remember, we used it for environment and collection. Builder and Team Library switches between Request Builder and Team Library views. Builder pane is tabbed. Top half is Request Builder and bottom half is Response

Viewer (like a console)

There's Cookies and Generate Code Snippets. Postman's native apps provide a MANAGE COOKIES modal that lets you edit cookies that are associated with each domain. The Cookies link is under the Send button in the Request Builder. I am not sure if we need cookies at this point.

https://www.getpostman.com/docs/postman/sending_api_requests/cookies

Now for the "onboarding" document. It was one of the links.

Documentation, Security and Credentials (see postman files)

Documentation:

- https://ie-cities-docs.run.aws-usw02-pr.ice.predix.io

Websocket URL:

- wss://ic-websocket-server.run.aws-usw02-pr.ice.predix.io/events

User Account & Authentication (UAA) Service (OAuth2):

- baseurl = https://890407d7-e617-4d70-985f-01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io
- Access token URI: {baseurl}/oauth/token?grant_type=client_credentials

Client ID / Client Secret:

- hackathon / @hackathon

Simple curl command to get a token:

- Base 64 encoded hackathon:@hackathon combination → aGFja2F0aG9uOkBoYWNrYXRob24=
- curl 'https://890407d7-e617-4d70-985f-01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token' -H 'Pragma: no-cache' -H 'content-type: application/x-www-form-urlencoded' -H 'Cache-Control: no-cache' -H 'authorization: Basic aGFja2F0aG9uOkBoYWNrYXRob24=' --data 'client_id=hackathon&grant_type=client_credentials'

current powered by GE

Documentation

"The CityIQ™ IoT Platform, powered by Predix, provides APIs to facilitate application development. The CityIQ APIs are part of the Intelligent Cities product group.

The main IoT edge device on this platform is a highly connected digital fabric of LED-VLC lights. The addition of sensors and controls technologies enables the LED-VLC lighting system to become an all-sensing network, compiling data for analytics and insights to drive productivity and efficiency. It is easily

extended by adding additional BLE sensors, OTS sensors, smart phones, and other edge devices."

This tells us who and what these APIs are designed to serve and be used by.

CityIQ is the set of APIs. Using them in the Hackathon is required to get a prize.

Predix "powers" the APIs.

What is Predix?

From Wikipedia,

> "Predix is General Electric's software platform for the collection and analysis of data from industrial machines.
>
> General Electric plans to support the growing industrial internet of things with cloud servers and an app store. GE is a member of the Industrial Internet Consortium which works to aid the development and use of industrial internet technologies.
>
> Predix as a cloud-based PaaS (platform as a service) is claimed to enable industrial-scale analytics for asset performance management (APM) and operations optimization by providing a standard way to connect machines, data, and people. GE expects Predix software to do for factories and plants what Apple's iOS did for cell phones. Built on Cloud Foundry open source technology, Predix provides a microservices based delivery model with a distributed architecture (cloud, and on-machine)."

They very very much want an app store just like Apple's.

It's very very unclear how app developers get paid in this type of ecosystem. The data in question here is "owned" by the City of San Diego, which has an immediete effect on monetization.

The Hackathon is supposed to spark ideas and applications that can become apps. An asset is a physical object, which has the capability to collect or exchange data. An asset can be a single

device (e.g. environmental sensor, camera, microphone), or a node (parent asset), which groups together multiple devices.

OK in the Getting Started of the Documentation section, “Each of the APIs is connected to a JAVA app data simulator, which simulates Cities and Enterprises activity from the node(s). You will first need to obtain a client account before you can start developing.”

Contact your “Current, Powered by GE” representative to obtain a client account, which includes the following, before you start developing:

1) client-id and client-secret to generate client tokens

client-id is hackathon and client-secret is @hackathon

2) predix-zone-id for each service

no idea what a predix-zone-id is yet

However, this:

baseurl = https://890407d7-e617-4d70-985f-01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io

and this:

Access token URI: {baseurl}/oauth/token?grant_type=client_credentials

Is a User Account & Authentication Service link and token.

3) service URLs

Simple curl command to get a token: •Base 64 encodedhackathon:@ hackathoncombination àaGFja2F0aG9uOkBoYWNrYXRob24=•curl ‘https://890407d7-e617-4d70-985f-01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token’-H ‘Pragma: no-cache’ -H ‘content-type: application/x-www-form-urlencoded’ -H ‘Cache-Control: no-cache’ -H ‘authorization: Basic aGFja2F0aG9uOkBoYWNrYXRob24=’ --data ‘client_id=hackathon&grant_type=client_credentials’

And... that's it for the onboarding sheet. If the information on the onboarding sheet doesn't require me to be at the Hackathon site, then today can be somewhat more productive.

Step 2: Get Up and Running with Hackathon Data

"Simulated Data Stream Information
GE Current recommends using the APIs to gather data and run the data simulator application prior to using real data.

Once you have obtained a client account, you can access the data simulator and retrieve simulated data and all relevant parameters in the Intelligent Cities APIs section: Appendix A: Simulated Data for Intelligent Cities Data Simulator."

OK going to Appendix A. Whoa! Data! It's a test environment. They warn me again I need system access. So I am going to test that right away. Nothing looks obvious so I decided to try the Runner button. It opens up in a new window. I selected the IC-VPC-APIs-Hackathon Collection and set the Environment to IC-VPC-APIs-Hackathon and left everything else default. I clicked Start Run.

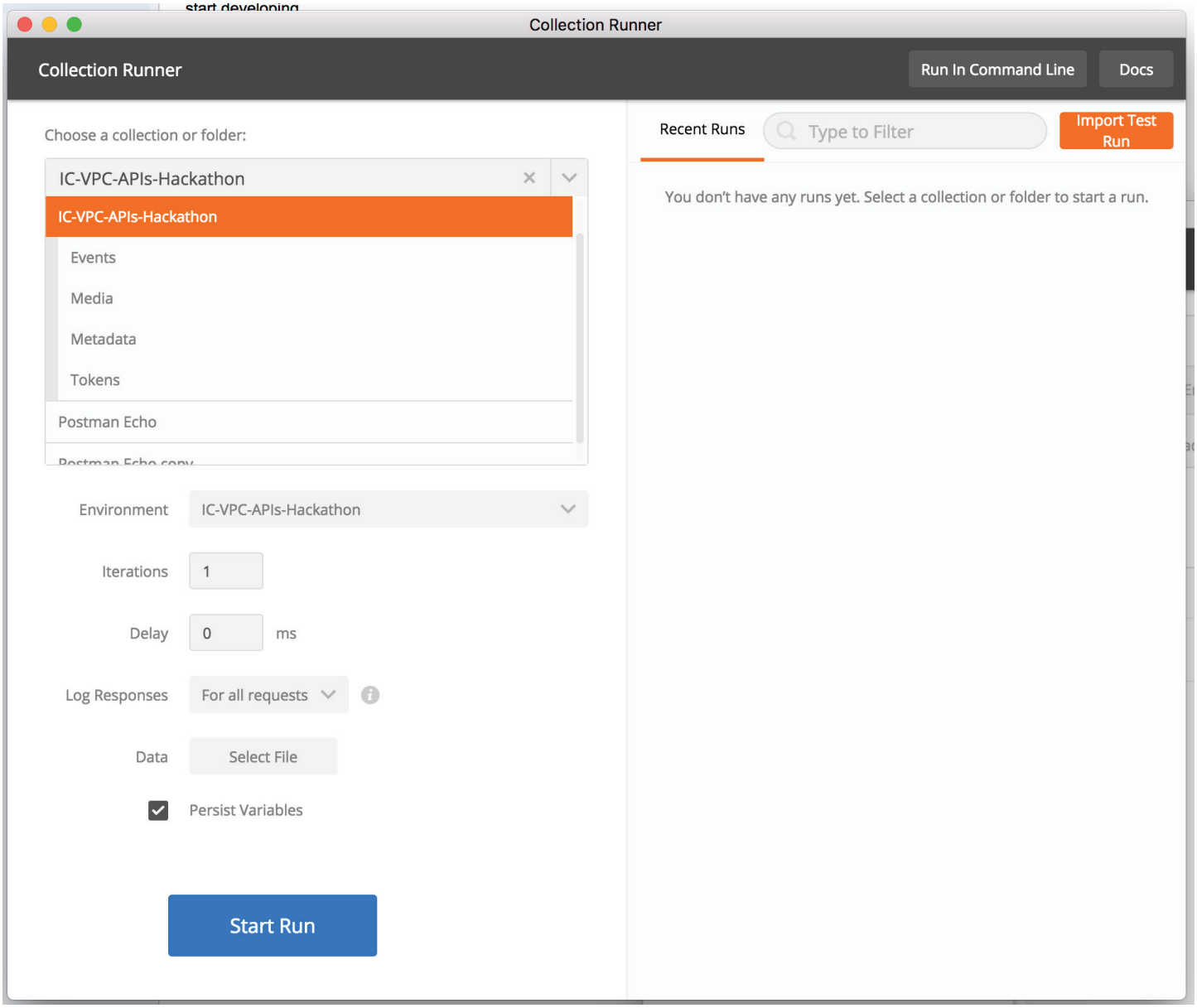
The Collection Runner seemed to work the first time. It ran the entire collection. It's probably using default or empty Events, Media, Metadata, Tokens, etc.

The other settings in Run Collection are Iterations. This is number of times the Collection will be run. Delay is the interval in milliseconds between each request. Log Response allows some granularity in logging. Data is data files supplied to the collection run. Example data file and how to use it here:

https://www.getpostman.com/docs/postman/collection_runs/working_with_data_files

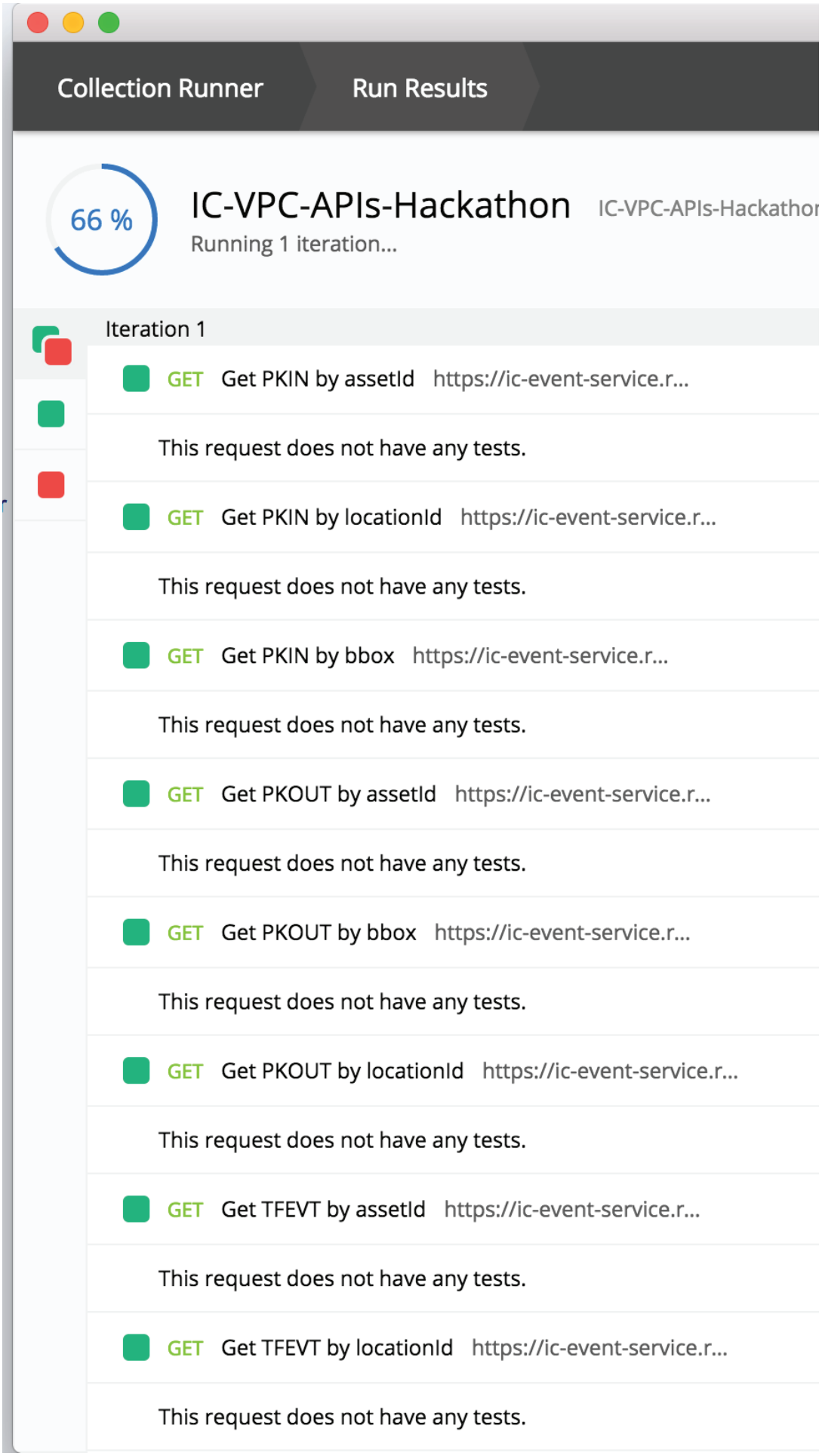
Persist Variable: "By default, any environment changes in the collection runner are not reflected back in the request builder" The variables in the Environment have four scopes. Global, Environment, Local, and Data.

Data scope is only available in the Collection Runner. When you Run a Collection. This means that changes made during the test run are not made to the variable in the request builder (I assume that the request builder scope is Local Scope). If you acutally do want your test run to modify data, and that modification to show up in the main Postman app request builder pane editor thing, then you check the Persist Variables check box. Note: on my Collection Runner, this box was checked by default and not unchecked. That seems backwards from the documentation. Just try and double check this.



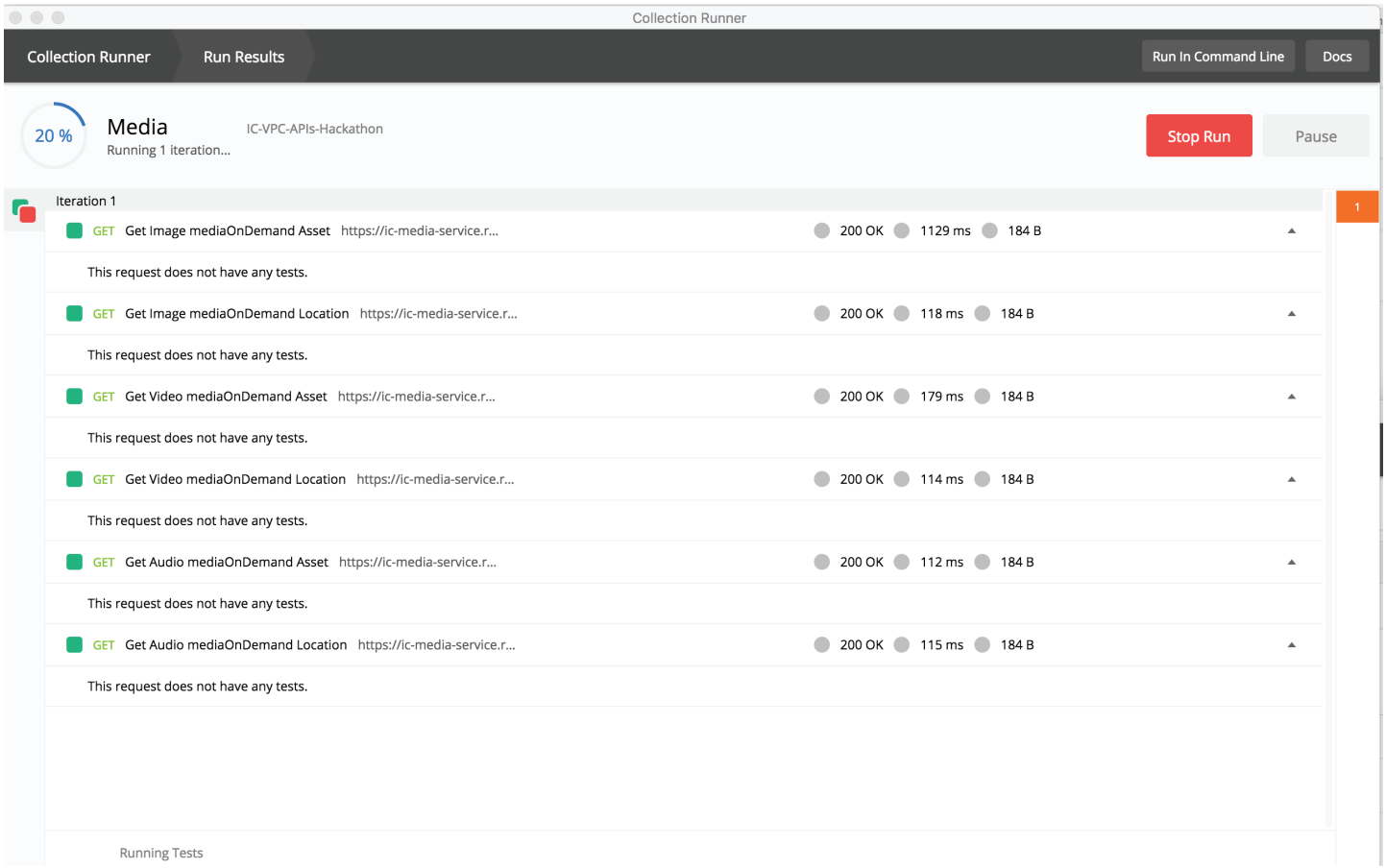
While the Collection Runner is running, it gives you a status of the Run. There's a Run results tab as well.

I selected Events from the Collection drop down in the Collection Runner. I ran the Collection and it completed.


Collection Runner
Run Results
66 %
IC-VPC-APIs-Hackathon
IC-VPC-APIs-Hackathor
Running 1 iteration...
Iteration 1
GET Get PKIN by assetId https://ic-event-service.r...
This request does not have any tests.
GET Get PKIN by locationId https://ic-event-service.r...
This request does not have any tests.
GET Get PKIN by bbox https://ic-event-service.r...
This request does not have any tests.
GET Get PKOUT by assetId https://ic-event-service.r...
This request does not have any tests.
GET Get PKOUT by bbox https://ic-event-service.r...
This request does not have any tests.
GET Get PKOUT by locationId https://ic-event-service.r...
This request does not have any tests.
GET Get TFEVT by assetId https://ic-event-service.r...
This request does not have any tests.
GET Get TFEVT by locationId https://ic-event-service.r...
This request does not have any tests.

I selected Media next, and it stalled at Get Audio mediaOnDemand Location. I know that there's a multi-step process for getting audio, so I suspect that the asynchronous media calling is sitting there waiting forever for the next step. I stopped the run after waiting a long time.

Next, Metadata was selected from the drop down in the Collection Runner. (don't forget to set environment as it forgets between runs which environment it was using)



The Run completed.

Finally, Tokens was selected. It had just one GET and it passed.

When you Export Results from the Run Results tab (for the Tokens Run), you get a Tokens.postman_test_run.json file that looks something like this:


```
{
    "id": "88f559dc-076d-440d-bebb-335ad3b2086a",
```

```
    “name”: “Tokens”,
    “allTests”: [],
    “timestamp”: “2017-06-22T17:07:46.676Z”,
    “collection_id”: “537c1756-4a3a-2f0d-42eb-c20a048e89b1”,
    “folder_id”: “92d4a7b1-5817-4ab6-2333-da1b7079da34”,
    “target_type”: “folder”,
    “environment_id”: “a0a3bac0-8d84-433d-ad02-41584f4d2845”,
    “data”: [],
    “delay”: 0,
    “count”: 1,
    “collection”: {
        “id”: “537c1756-4a3a-2f0d-42eb-c20a048e89b1”,
        “name”: “IC-VPC-APIs-Hackathon”,
        “description”: “”,
        “order”: [],
        “folders”: [
            {
                “id”: “5cb3a312-725e-3790-b34b-c3b3539a48cd”,
                “name”: “Events”,
                “description”: “”,
                “order”: [
                    “53f6561f-07d8-c36f-8917-1996f05a0b2b”,
                    “ca092e27-5fd0-08b3-215c-a68513eb466e”,
                    “82c0ada4-7849-14d4-4daa-4b2176d2ef56”,
                    “1a4f8d64-c1c1-901b-b653-878adf252e13”,
                    “191379ff-ce4b-fc41-b709-b6565f36cd65”,
                    “3010035a-e4d4-b4b6-5df6-b662d059e189”,
                    “8cc794a2-fab0-fcbe-bb7d-a408f0bee3a0”,
                    “bac031d4-38d2-7e4a-e2f1-65571a7cd3a4”,
                    “2659be7c-efdd-373a-a081-50965bc1d81a”,
                    “eb505c55-0776-c96e-740a-6c6a01e8c8d7”,
                    “ac351531-1ba1-8c3c-2213-45dc0d21842c”,
                    “6f631ce4-6521-5fb2-fef9-20cafe4734a0”,
                    “9600dc80-ff5a-1b90-fd26-5e906a5cfd28”,
                    “ed32d2cc-05d8-7d7e-2fae-22800a92f149”
                ],
                “owner”: “2276179”,
                “collection_id”: “537c1756-4a3a-2f0d-42eb-
c20a048e89b1”,
                “collection”: “537c1756-4a3a-2f0d-42eb-
c20a048e89b1”
            },
            {
```

```
                "id": "6314049d-0c1b-1215-90ca-8d07245131da",
                "name": "Media",
                "description": "",
                "order": [
                    "dd658515-035e-7efa-d35e-7a51475eec5d",
                    "939c9f14-b837-3689-71c0-2e396514a49f",
                    "c62825c8-3083-6eeb-562b-b7a3c67e7ed2",
                    "e1c53bd8-2559-2418-5ad1-355c33d1ddd2",
                    "ad718c83-80a3-871a-ddcb-fcc8ec7f290a",
                    "2aaa8fb3-6792-bff9-64a7-bbe1509f0fd4",
                    "f44c198f-ae9f-5d1d-52bf-6fece7951718",
                    "ea9b7a7d-7d0b-0898-8a41-78216d924b62"
                ],
                "owner": "2276179",
                "collectionId": "5e94bf7a-8c33-08f1-6704-
e79436b5f758",
                "collection_id": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1",
                "collection": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1"
            },
            {
                "id": "16b2bd28-bb00-a52d-f962-ea499b55d1fc",
                "name": "Metadata",
                "description": "",
                "order": [
                    "4573a4b2-9cfa-8029-db84-87b4a00cc22d",
                    "d3e3b583-3420-8ecd-f7e0-f5452756acf5",
                    "9c623125-55d4-6ca0-af55-8cc75d3dc559",
                    "c5d93603-6a42-44a4-299c-8b92feb96832",
                    "207af465-01e7-f61e-b14b-276d93454760",
                    "77d37655-b733-60c4-db24-5376062533c8",
                    "2892fa5d-d607-f6b5-83e3-955acc127362"
                ],
                "owner": "2276179",
                "collection_id": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1",
                "collection": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1"
            },
            {
                "id": "92d4a7b1-5817-4ab6-2333-da1b7079da34",
                "name": "Tokens",
```

```
                "description": "",
                "order": [
                    "b9ff228b-61ab-e7e7-e900-af16b105a542"
                ],
                "owner": "2276179",
                "collection_id": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1",
                "collection": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1"
            }
        ],
        "timestamp": 0,
        "synced": true,
        "remote_id": 0,
        "owner": "2276179",
        "sharedWithTeam": false,
        "subscribed": false,
        "remoteLink": "",
        "remoteLinkUpdatedAt": null,
        "public": false,
        "createdAt": 1498146875127,
        "updatedAt": 1498146875127,
        "write": true,
        "published": false,
        "favorite": false,
        "permissions": {},
        "syncedPermissions": {}
    },
    "folder": {
        "id": "92d4a7b1-5817-4ab6-2333-da1b7079da34",
        "name": "Tokens",
        "description": "",
        "order": [
            "b9ff228b-61ab-e7e7-e900-af16b105a542"
        ],
        "owner": "2276179",
        "collection_id": "537c1756-4a3a-2f0d-42eb-
c20a048e89b1",
        "collection": "537c1756-4a3a-2f0d-42eb-c20a048e89b1"
    },
    "environment": null,
    "globals": [],
    "results": [
```

```
        {
            “name”: “Get client Token”,
            “id”: “b9ff228b-61ab-e7e7-e900-af16b105a542”,
            “url”: “https://890407d7-e617-4d70-985f-
01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/
token?grant_type=client_credentials”,
            “totalTime”: 0,
            “responseCode”: {
                “code”: 200,
                “name”: “OK”,
                “detail”: {
                    “name”: “OK”,
                    “detail”: “Standard response for
successful HTTP requests. The actual response will depend on
the request method used. In a GET request, the response will
contain an entity corresponding to the requested resource. In a
POST request the response will contain an entity describing or
containing the result of the action.”
                }
            },
            “tests”: {},
            “testPassFailCounts”: {},
            “times”: [
                “841”
            ],
            “allTests”: [
                {}
            ],
            “time”: “841”,
            “totalRequestTime”: “841”,
            “iterationResults”: {}
        }
    ],
    “totalPass”: 0,
    “totalFail”: 0,
    “totalTime”: 841,
    “lifecycle”: “done”,
    “requests”: [
        {
            “name”: “Get client Token”,
            “id”: “b9ff228b-61ab-e7e7-e900-af16b105a542”,
            “url”: “https://890407d7-e617-4d70-985f-
01792d693387.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/
```

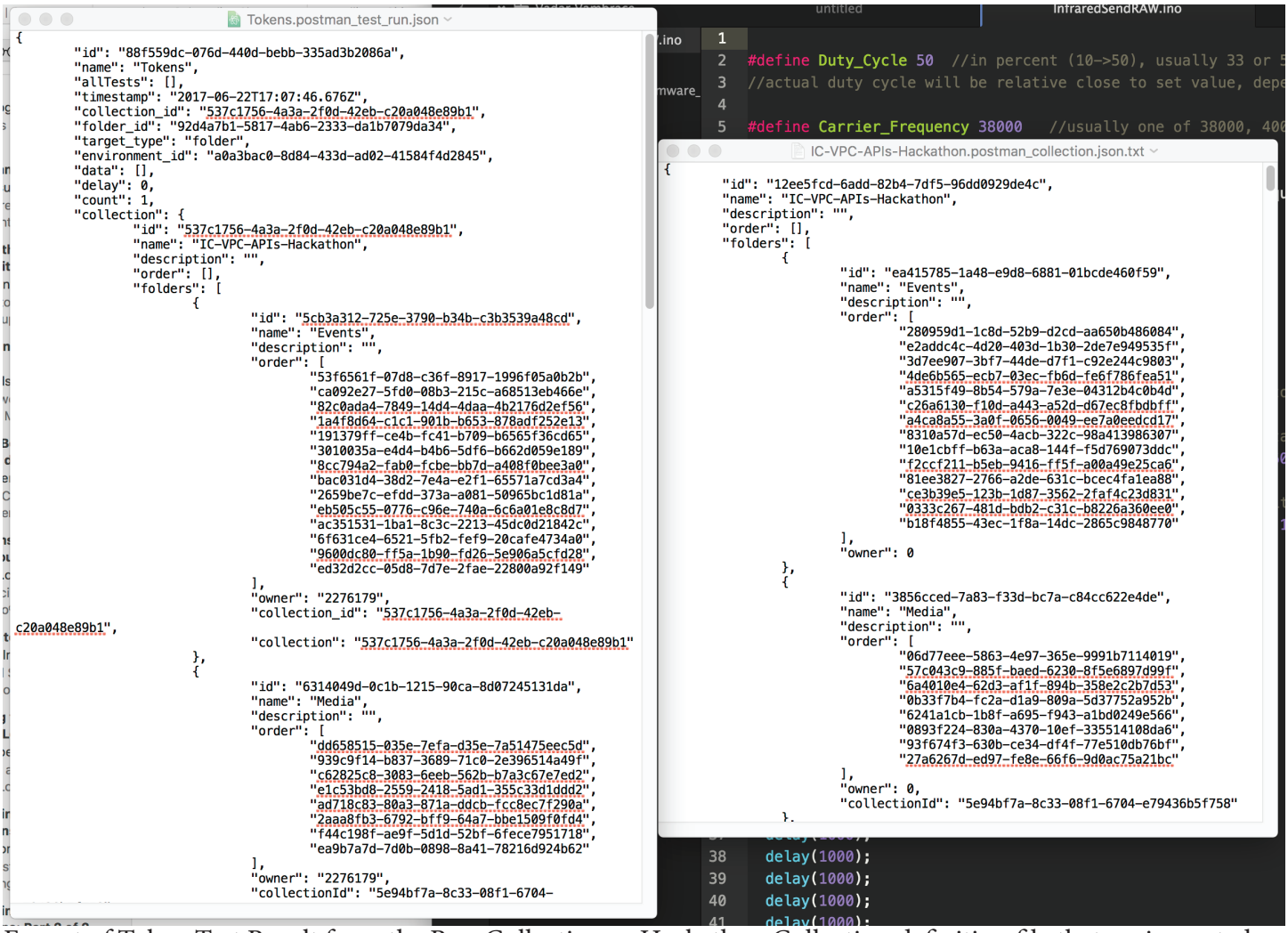
```
token?grant_type=client_credentials",
            "time": "2017-06-22T17:07:46.676Z",
            "responseCode": {
                "code": 200,
                "name": "OK",
                "detail": {
                    "name": "OK",
                    "detail": "Standard response for
successful HTTP requests. The actual response will depend on
the request method used. In a GET request, the response will
contain an entity corresponding to the requested resource. In a
POST request the response will contain an entity describing or
containing the result of the action."
                }
            },
            "testPassFailCounts": {}
        }
    ],
    "synced": false
}
```


Well! That's a lot of JSON! Our particular run was for Tokens, but the first majority of the file is test run results for the other categories (Events, Media, and Metadata). The Run Collection seems to be picking up the right Collection and Environment information from the Collection and Environment JSON imports, and is using the settings from the Run Collection settings box. Comparing the two files shows the same folder structure (see next page).

Now we need to set up a real Token fetch. I think the Tokens are client tokens (that's what they're called) and I think they are important.

While looking for this, I found a very simple Collection and Environment that allowed for a very simple Hello World experiment that also showed the Data Scope functionality. That experiment is here:

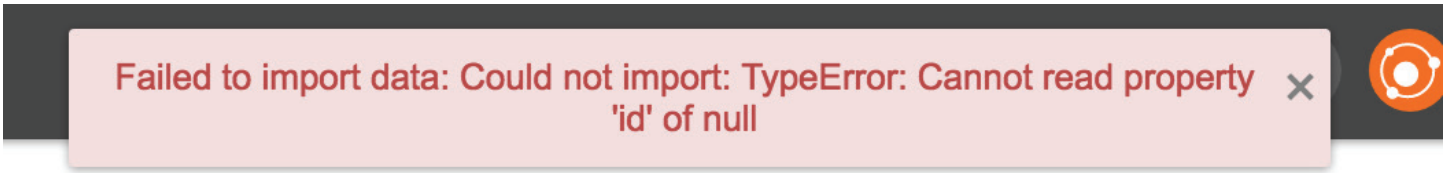
https://www.getpostman.com/docs/postman/collection_runs/using_environments_in_collection_runs



Export of Token Test Result from the Run Collection vs Hackathon Collection definition file that we imported.

I went ahead and tried it and got the same import error as I got with the Hackathon files. This time I made a screen shot. Despite the scary Failed to import data message, the error doesn't appear to negatively affect Postman operation. Yet.

So, I ran the one pre-fabbed POST post-man-echo.com/{{path}}



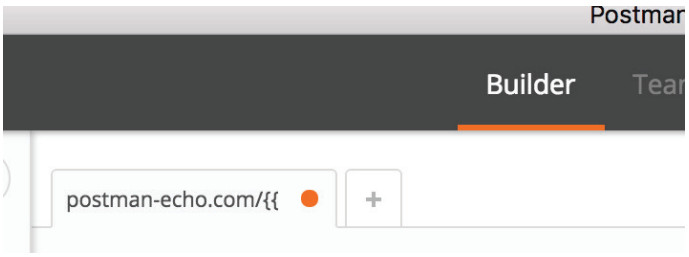
request and got the results expected. Manually modifying the modified variables in the Environment worked as expected, as well. I wasn't entirely sure what I was doing, so I read up.

Basics of POST post-man-echo.com/{{path}}

POST, etc (see at right) are HTTP “verbs”. The basic idea here is a REST API, or Representational State Transfer. REST is an architecture style for designing networked applications. “REST relies on a stateless, client-server, cacheable communications protocol.” The HTTP protocol is used.

Postman uses the {{handlebars}} format for all environment variables. You can use environment variables in any text entry field in the resource pane.

So now I could find both halves of my derriere with my hands, I tried to add a basic GET request to the Hello World test Collection and Environment from Postman. I opened up a new tab in the Builder plane by clicking the plus tab marker (above).

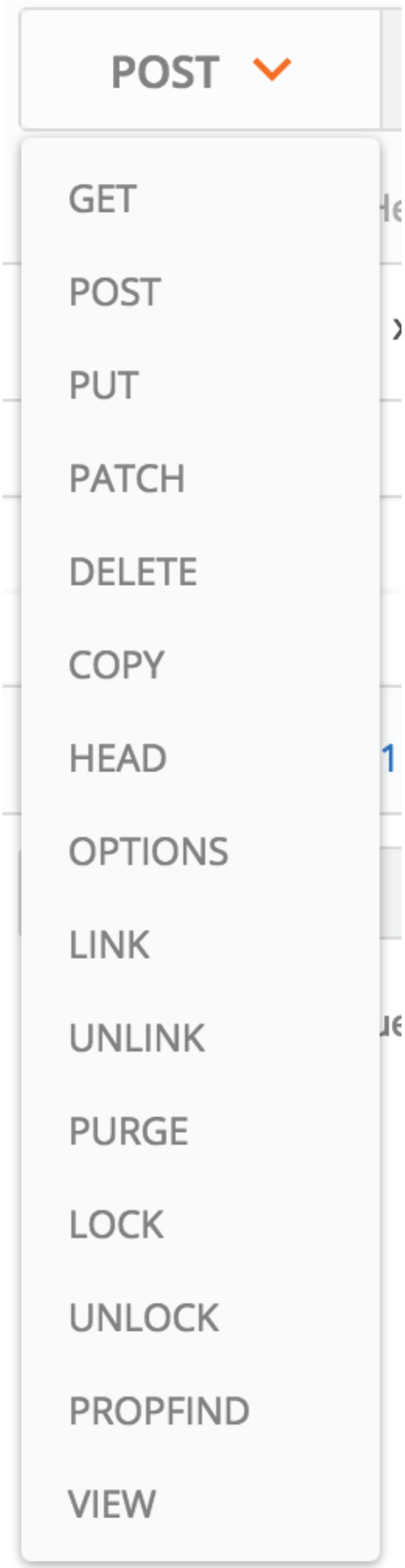


After that, I selected GET from the drop down and put the below URL from a tutorial in to the “Enter request URL” box

http://jsonplaceholder.typicode.com/posts

Hit send, and received a fun Lorem Ipsum JSON file back. OK so I can POST and GET.

I saved the new request to the Using Environments Collection and it worked. I created a folder and moved the two requests into the new folder just to make sure I knew how that worked. You can create a folder and then drag and drop. I then

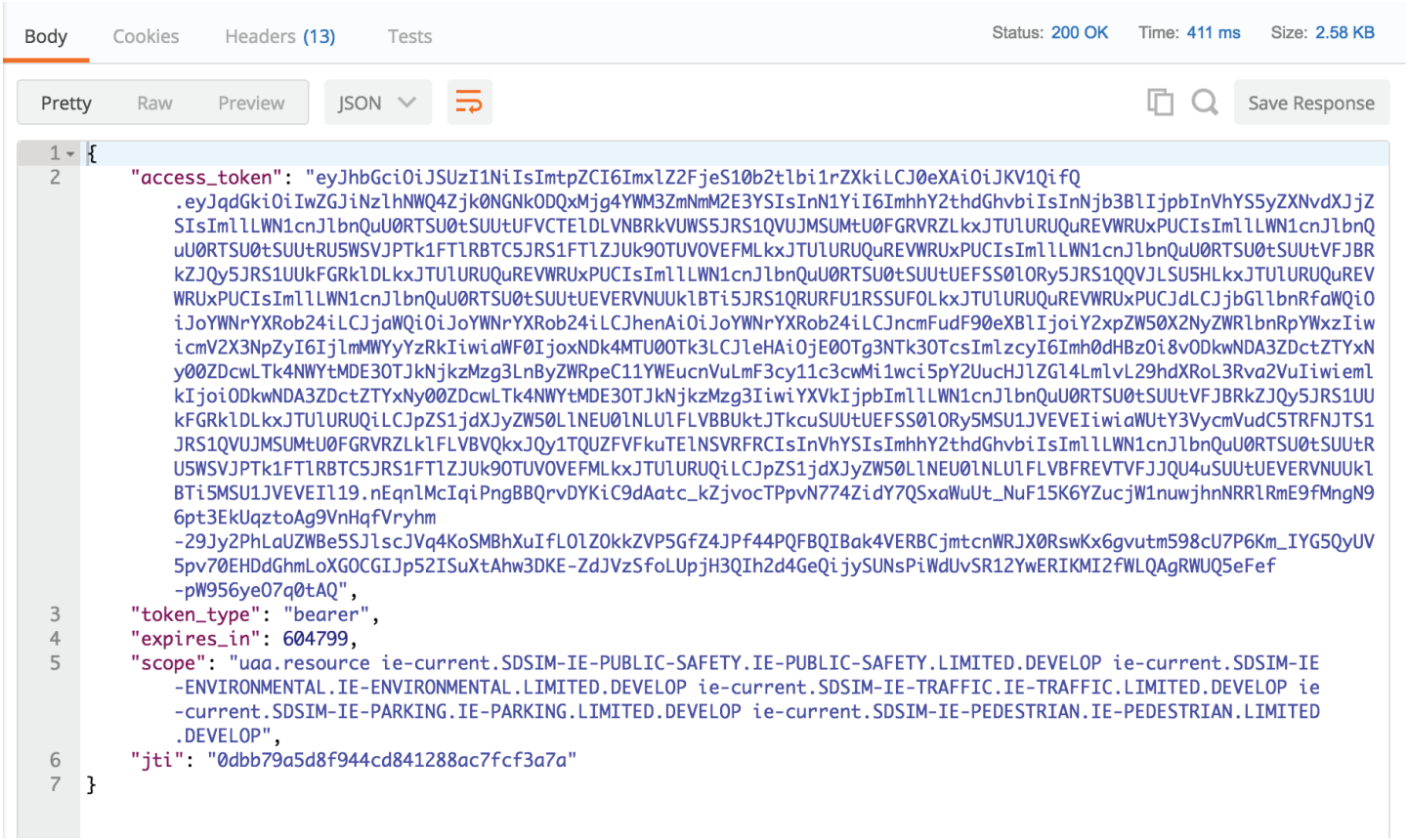


Here’s all the choices for requests in the request builder.

could go to Collection Runner and select that particular folder. The power of folders in collections seems to be that you can then select a folder and run just the requests in that folder in the Collection Runner. It allows separating out reuests for runs. Sure enough, turning off Persist Variables results in the POST shennanigans passing the test even though the POST test modifies the foo bar key value pair. OK!

Now what? Time to try to use the Hackathon Collection and Environment?

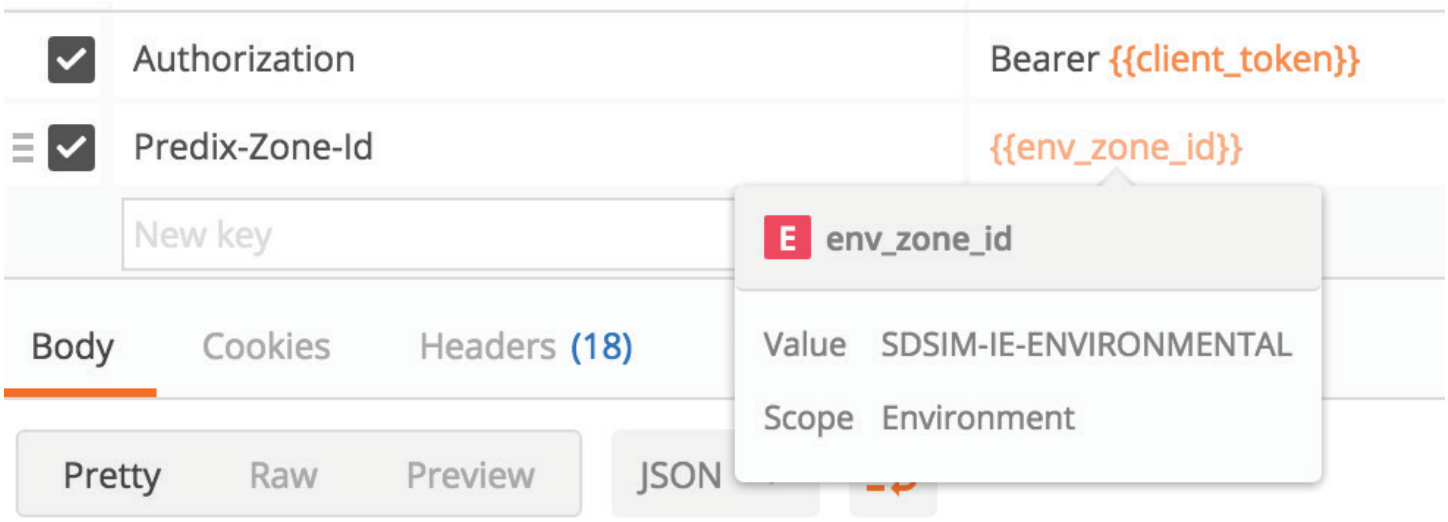
I tried the get client token and it seemed to work. I don't know what to do with the token, so I moved on to Get TEMPERATURE by assetid. Results from sending the get client token request are below.
The parameters associated with Get TEMPERATURE by assetid are

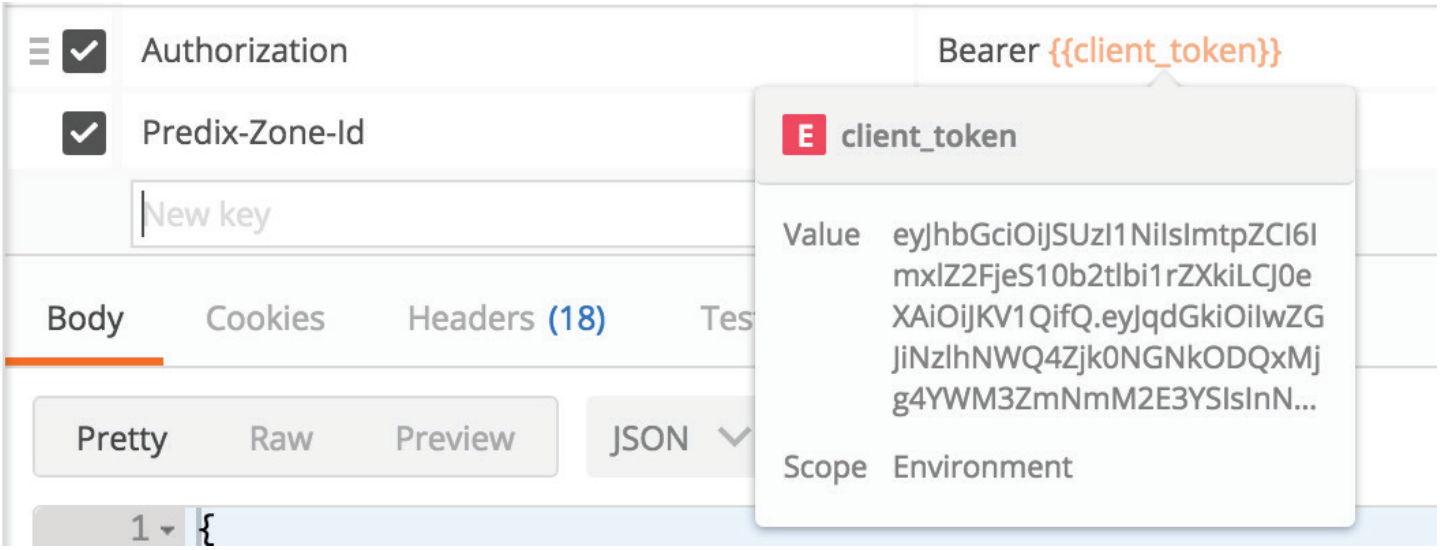



```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImxlZ2FjeS10b2tlbi1rZXkiLCJ0eXAiOiJKV1QifQ
        .eyJqdGkiOiIwZGJiNzlhNWQ4Zjk0NGNkODQxMjg4YWM3ZmNmM2E3YSIsInN1YiI6ImhhY2thdGhvbiIsInNjb3BlIjpbInVhYS5yZXNvdXJjZ
        SIsImlllWN1cnJlbnQuU0RTSU0tSUUtUFVCTElDLVNBRkVUWS5JRS1QVUJMSUMtU0FGRVRZLkxJTUlURUQuREVWRUxPUCIsImlllWN1cnJlbnQ
        uU0RTSU0tSUUtRU5WSVJPTk1FTlRBTC5JRS1FTlZJUk9OTUVOVEFMLkxJTUlURUQuREVWRUxPUCIsImlllWN1cnJlbnQuU0RTSU0tSUUtVFJBR
        kZJQy5JRS1UUkFGRklDLkxJTUlURUQuREVWRUxPUCIsImlllWN1cnJlbnQuU0RTSU0tSUUtUEFSS0lORy5JRS1QQVJLSU5HLkxJTUlURUQuREV
        WRUxPUCIsImlllWN1cnJlbnQuU0RTSU0tSUUtUEVERVNUUklBTi5JRS1QRURFU1RSSUFOLkxJTUlURUQuREVWRUxPUCJdLCJjbGllbnRfaWQiO
        iJoYWNrYXRob24iLCJjaWQiOiJoYWNrYXRob24iLCJhenAiOiJoYWNrYXRob24iLCJncmFudF90eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIiw
        icmV2X3NpZyI6IjlmMWYyYzRkIiwiaWF0IjoxNDk4MTU0OTk3LCJleHAiOjE0OTg3NTk3OTcsImlzcyI6Imh0dHBzOi8vODkwNDA3ZDctZTYxN
        y00ZDcwLTk4NWYtMDE3OTJkNjkzMzg3LnByZWRpeC11YWEucnVuLmF3cy11c3cwMi1wci5pY2UucHJlZGl4LmlvL29hdXRoL3Rva2VuIiwieml
        kIjoiODkwNDA3ZDctZTYxNy00ZDcwLTk4NWYtMDE3OTJkNjkzMzg3IiwiYXVkIjpbImlllWN1cnJlbnQuU0RTSU0tSUUtVFJBRkZJQy5JRS1UU
        kFGRklDLkxJTUlURUQiLCJpZS1jdXJyZW50LlNEU0lNLUlFLVBBUktJTkcuSUUtUEFSS0lORy5MSU1JVEVEIiwiaWUtY3VycmVudC5TRFNJTS1
        JRS1QVUJMSUMtU0FGRVRZLklFLVBVQkxJQy1TQUZFVFkuTElNSVRFRCIsInVhYSIsImhhY2thdGhvbiIsImlllWN1cnJlbnQuU0RTSU0tSUUtR
        U5WSVJPTk1FTlRBTC5JRS1FTlZJUk9OTUVOVEFMLkxJTUlURUQiLCJpZS1jdXJyZW50LlNEU0lNLUlFLVBFREVTVFJJQU4uSUUtUEVERVNUUkl
        BTi5MSU1JVEVEIl19.nEqnlMcIqiPngBBQrvDYKiC9dAatc_kZjvocTPpvN774ZidY7QSxaWuUt_NuF15K6YZucjW1nuwjhnNRRlRmE9fMngN9
        6pt3EkUqztoAg9VnHqfVryhm
        -29Jy2PhLaUZWBe5SJlscJVq4KoSMBhXuIfLOlZOkkZVP5GfZ4JPf44PQFBQIBak4VERBCjmtcnWRJX0RswKx6gvutm598cU7P6Km_IYG5QyUV
        5pv70EHDdGhmLoXGOCGIJp52ISuXtAhw3DKE-ZdJVzSfoLUpjH3QIh2d4GeQijySUNsPiWdUvSR12YwERIKMI2fWLQAgRWUQ5eFef
        -pW956yeO7q0tAQ",
    "token_type": "bearer",
    "expires_in": 604799,
    "scope": "uaa.resource ie-current.SDSIM-IE-PUBLIC-SAFETY.IE-PUBLIC-SAFETY.LIMITED.DEVELOP ie-current.SDSIM-IE
        -ENVIRONMENTAL.IE-ENVIRONMENTAL.LIMITED.DEVELOP ie-current.SDSIM-IE-TRAFFIC.IE-TRAFFIC.LIMITED.DEVELOP ie
        -current.SDSIM-IE-PARKING.IE-PARKING.LIMITED.DEVELOP ie-current.SDSIM-IE-PEDESTRIAN.IE-PEDESTRIAN.LIMITED
        .DEVELOP",
    "jti": "0dbb79a5d8f944cd841288ac7fcf3a7a"
}
```


Authorization and Predix-Zone-Id. Remember that the onboarding said we needed one? It seems to me that the Zone ID is provided by the Environment file. The Authorization is I hope {{client_token}}, which I ran immediately before. Scope is also Environment. Maybe the get client token request wrote the client token to the Environment file? I went looking for it in the Environment, and sure enough - client token from the request was

saved to the Environment JSON file. client_token key and the same value from the request result was the value. OK good so far.





env_zone_id was present in the environment as a key and the value in the environment file matched what was in the request builder. Where did the Predix-Zone-Id and Authorization come from? What are they doing? If you give the Predix-Zone-Id the right key, then things work. Predix-Zone-Id is a “parameter” in the request builder. Each API has a unique XXX_zone_id that I assume makes the Predix-Zone-Id parameter happy. So, just match the zone ID to the API when a request is made.

Authorization is Bearer {{client_token}}. If you change from No Auth to Basic Auth, there’s a chance that the value in Authorization in the Header will be changed to a static value.

When this happened to me through experimenting, authorization failed, and even changing back to No Auth didn't fix it. Manually editing the Authorization field back to Bearer {{client_token}} was required.

Next I added a new key:value pair into the environment file. Start and end times were listed as variables, so I made a new time in between. Using the new time as the end time reduced the amount of data recieved in response. Good!

Variables

Hovering over a variable shows its current value and the scope.

Often while using variables in scripts, you will need to see the values they obtain. You can use the Postman Console to do this easily. From the application menu, select "View" and then "Show Postman Console". To log the value of a variable, you can use console.log(foo); in your script. When you send a request, the script will be evaluated and the value of the variable will be logged in the Postman Console.

Variables are highlighted in orange, with unresolved variables shown in red colour.

Type an open curly bracket to bring up the autocomplete menu.

The "eye"con in the upper right opens up the environment variable file and you can edit from here too.

Postman has some dynamic variables. You can use in your requests. You can only use them in the {{..}} format in the request URL / headers / body.

{{$guid}} : Adds a v4 style guid
{{$timestamp}}: Adds the current timestamp
{{$randomInt}}: Adds a random integer between 0 and 1000

Data Variables, used most often in Collection Runner, are from a data file that is imported into the Collection Runner. This data file is optional. This file is not the same as the environment variables file. Data file variables operate at the Data scope, which is below Environment. This is related to the Persist

Variables checkbox in the Collection Runner.

You can use these data variables inside Postman with the same syntax as environment or global variables, with the {{variable}} syntax.

Inside pre-request and test scripts, there is a special data object. It contains values loaded from the data file for a specific iteration.

data.username
data[“username”]

both let you access the value of the username variable from a data file, when you are in pre-request and test scripts.

Variables are often used in pre-request and test scripts. “Since these sections for scripts are written in JavaScript, you will initialize and retrieve these variables in a different manner.” You can initialize variables in the JavaScript scripts and put them in a particular scope.

Defining a variable in a script:
To set a variable in a script, use the setEnvironmentVariable() method or setGlobalVariable() method depending on the desired scope. The method requires the variable key and value as parameters to set the variable. When you send the request, the script will be evaluated and the value will be stored as the variable.

Fetching a pre-defined variable:
Once a variable has been set, use the getEnvironmentVariable() method or getGlobalVariable() method depending on the appropriate scope to fetch the variable. The method requires the variable name as a parameter to retrieve the stored value in a script.

Setting a variable in a scope:
Environment variables can be accessed with the corresponding environment template - make sure you have it loaded in the drop-down menu in the upper right. Global variables can be accessed broadly regardless of the selected environment.

Test Scripts

The Test script window lets you write JavaScript test scripts. These are automated tests that run on the requests.

http://blog.getpostman.com/2014/03/07/writing-automated-tests-for-apis-using-postman/

Part 2 of the above blog adds a bit more functionality.

WebSocket Service

Use the WebSocket service to receive a stream of near real-time data for an asset or assets in a location.

Access the provided URI to open a TCP connection to enable two-way communication between your client and the service, and specify the data to be received with a JSON formatted request.

“Live” functions request and receive near real-time data using WbSocket, and “filter” funtions retrieve event data using HTTPS.

You need to send the bearer token and Predix-Zone-Id while establishing the Websocket connection!

I attempted to use the Intelligent Cities API to construct a real-time WebSocket to get Temperature from a particulare asset. It’s the first entry at:

https://ie-cities-docs.run.aws-usw02-pr.ice.predix.io/#r_get_environmental_planning_api.html

And ended up with:

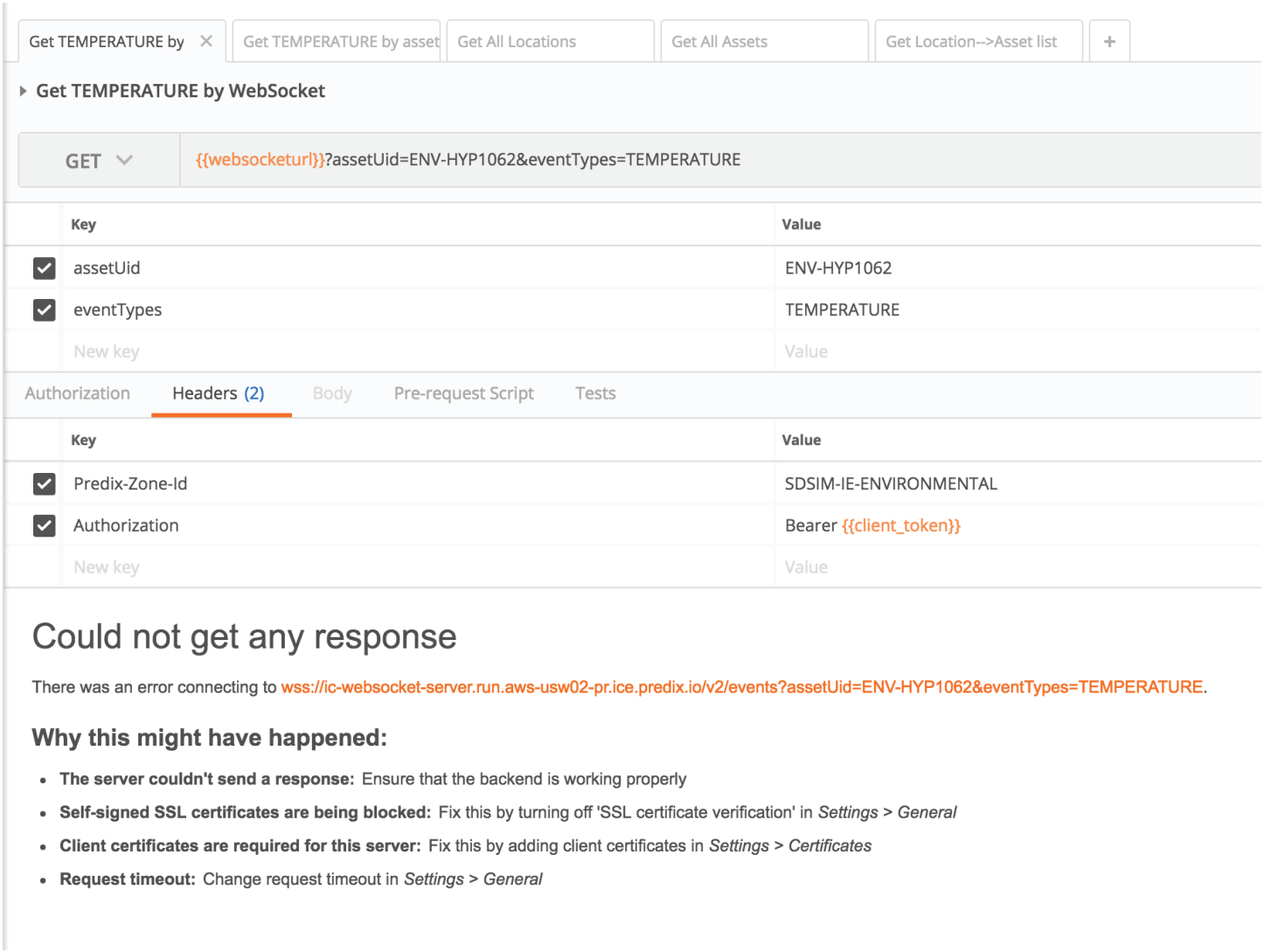
GET {{websocketurl}}?assetUid=ENV-HYP1071&eventTypes=TEMPERATURE

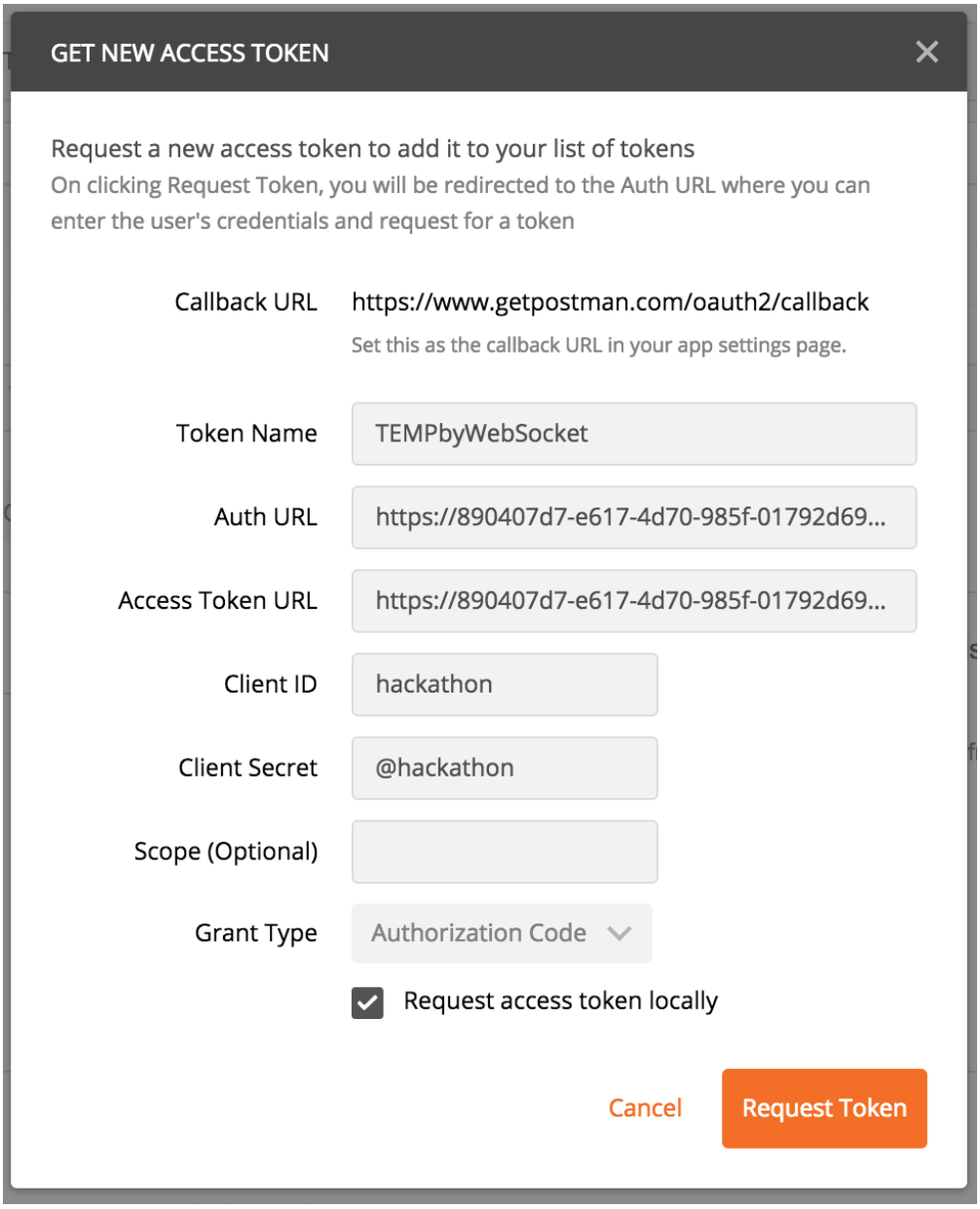
The asset ID was from the provided historical TEMPERATURE Get by assetID, and shows up in the “get assets” results as well. So I think that the assetUid is correct. I know eventTypes = TEMPERATURE is correct. I tried with No Auth and it didn’t work. I turned on Basic Auth and it didn’t work. Basic Auth changed the Authorization Header (as expected) from Bearer {{client_

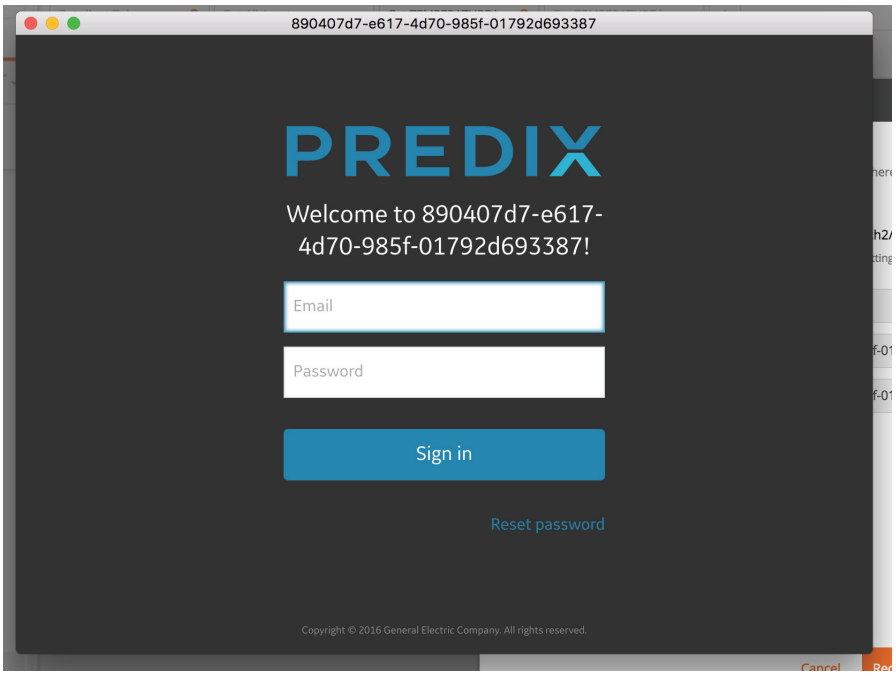
token}} to Basic aGFja2F0aG9uOkBoYWNrYXRob24=

I changed it back to Bearer {{client_token}} and tried OAuth2.

That didn't work either. "Could not complete Oauth2 login" from popup window after "get new access token" dialogue box was filled out. Maybe the problem is OAuth2 or maybe the probel is a bad request. Not sure which!




GET NEW ACCESS TOKEN
Request a new access token to add it to your list of tokens
On clicking Request Token, you will be redirected to the Auth URL where you can enter the user's credentials and request for a token
Callback URL
https://www.getpostman.com/oauth2/callback
Set this as the callback URL in your app settings page.
Token Name
TEMPbyWebSocket
Auth URL
https://890407d7-e617-4d70-985f-01792d69...
Access Token URL
https://890407d7-e617-4d70-985f-01792d69...
Client ID
hackathon
Client Secret
@hackathon
Scope (Optional)
Grant Type
Authorization Code
Request access token locally
Cancel
Request Token


890407d7-e617-4d70-985f-01792d693387
PREDIX
Welcome to 890407d7-e617-4d70-985f-01792d693387!
Email
Password
Sign in
Reset password
Copyright © 2016 General Electric Company. All rights reserved.