Rationale and Design

John Singer

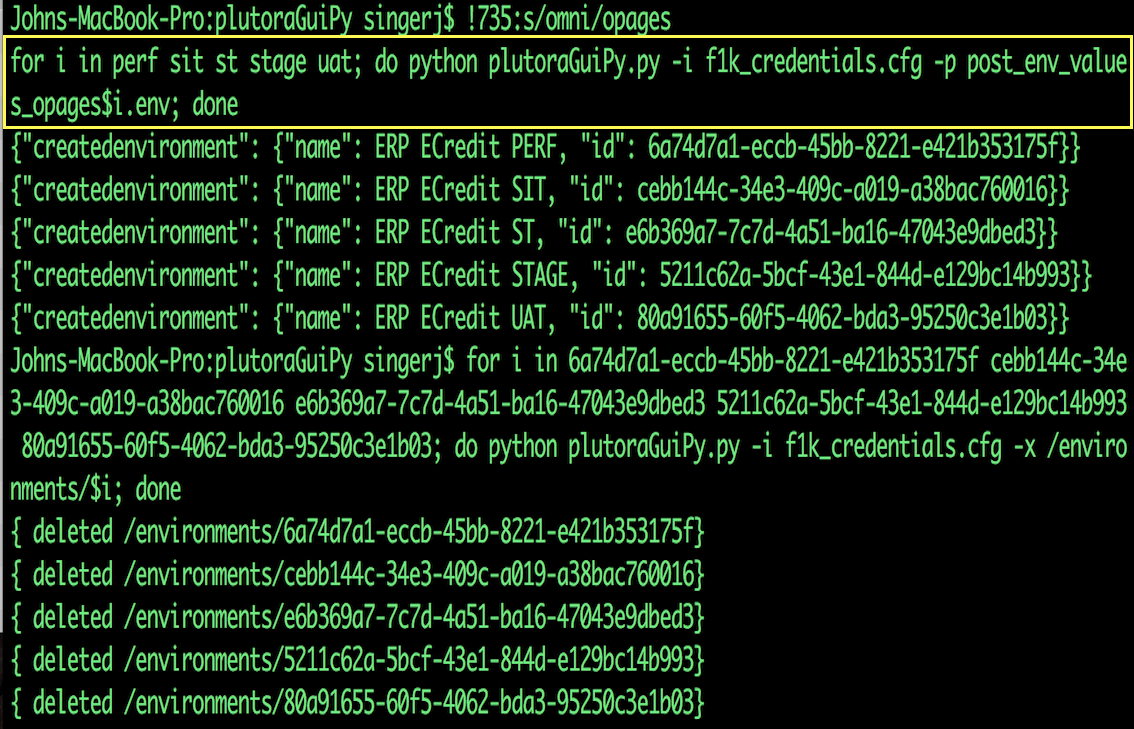
PlutoraGuiPy - Making Life Easier

February 16, 2017

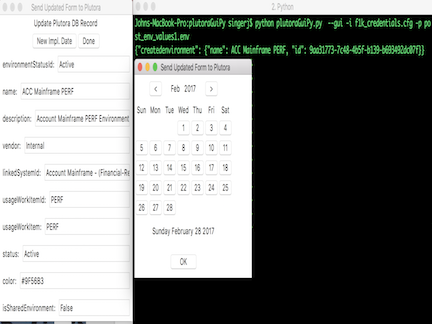
PlutoraGuiPy: Rationale, Design, & Thinking

## Background:

The first question is obviously ‘Why?’. Why write PlutoraGuiPy? Why in Python? Why not just use the existing SaaS GUI? Well, the milieu; and the history, you know.

When I ‘came on’ as a Solutions Consultant at the end of December, 2016, I was supposed to demonstrate, install, implement, and orchestrate customers Plutora installations. Also, we had some ‘demo site’ problems: Once laboriously set up, the demos became ‘stale’ as time went on. Release management is all about managing complexity over time, you know.   
The obvious solution was to create ‘automation’. And it was something I could do ‘on my own’ as I waited for human resources to ‘come online’, after the holidays.

Partially because of some of the work I did at IBM, and partially because I’ve programmed in 68 different languages, I was interested in building tools; ones that made my life easier. And I figured I’d write about the process so other newbies and customers could ‘learn from my missteps’ (which, it turned out, were myriad; but I think typical of the way a customer-engineer might approach this). So here is my ‘*chronologue'*

As a programer of 20 years, what better way to learn the “nooks and crannies” than to create automation around it? There was a ReST API, so ‘why not?’. And Python was the logical choice because most everyone I’ve met can either learn Python or understand it. And it comes installed on our Macs. And it has a REPL. And there are plenty of Python-flavored IDEs with debugging facilities ‘out there’. And so I started…

“What if I just need to ‘tweak’ one thing, though?

-Sales Guy

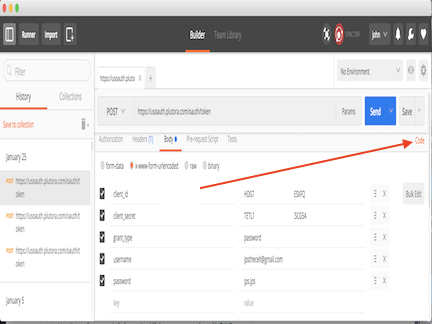
I initially wanted to perform the simplest repetitive task I could think of: ‘create a release’ — wrong idea. It turned out that there were a lot of ‘interlocking’ pieces of information I needed to know and ‘get right’.

OK, so, I figured I’d try to create an environment; wrong, again. Same problem. So, I went to the API help page (<https://usapi.plutora.com/Help/Api>), looking for their **swagger** — no such luck. So looked through the help pages for the simplest thing I could create — it turned out to be a ‘system’. Five fields and three of them were strings. OK, ‘I got this’.

‘Not so fast, cowboy!’ First, you’ve got to get ‘authenticated’, and then "find ‘organizational guid’, you must”. Hmmmm; starting to sound like a bad dream starring ‘Yoda’.

I wanted to see someone’s code who’d done this, before — I looked around GitHub and Google & found absolutely nothing; sigh.

I searched the Plutora-site and found Charlotte’s ‘Getting started with the Plutora API’. OK, that helped. I then went to the ‘company’ Plutora Customization/Settings/API section, enabled it, got my **client\_id** and **client\_secret**, installed and used Postman, got my **authorization-token**, and was able to retrieve my email-address. WHEW! At least I can read something from the tool.

Using my ‘hard-won’ knowledge (and the excellent python **requests** library), I set about trying to create a program that would merely reproduce what I’d done ‘by hand’. After ‘flailing’ a bit, I discovered that Postman has a neat feature where, once you’ve done something in the tool, you can click on the ‘produce code in X, for me’ button. Wow! I never noticed THAT, before; cool!:

So I selected ‘Python’, copied the code, and ran it successfully.

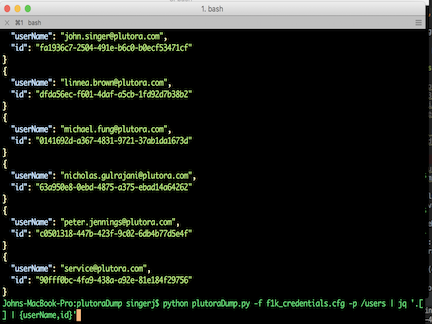
“Well, I said”, backing up, mentally; maybe I shouldn’t try to first POST stuff; maybe I should write a ‘data-dump’ utility (see <http://github.com/plutoraDump.git> ). The main reason for this was to enable me to get the afore-mentioned **GUIDs** (see ‘id’, in the above pict).

This utility has been through a few iterations in the past couple of months, but the concept remains the same: dump the indicated data-structure in JSON, so you can look at it and learn. It’s a command-line utility, so I can use the repetitive capabilities of Bash ‘for’ loops and substitutions (I put my username, password, client\_id, and client\_secret in the JSON-file **f1k\_credentials.cfg**, and the name of the ‘sub-URL’ I want to query, after the **-p** .

I also modified the tool to display JSON-formatted output, so I can ‘pipe’ it through the utility JSON-query (**jq** — <https://stedolan.github.io/jq/> ). This enables ‘command-chains’ like the following:

**python plutoraDump.py -f f1k\_credentials.cfg -p /releases | jq '.[] | {username,id}'**

(this is the same ‘dump’ command, this time piped through **jq.** It not only selects just the **userName** and **id** fields, but ‘pretty-prints’ them in colors).

Ok, at any rate I was able to get the name and GUId for an organization, so I could now create a Plutora system. My resultant code is in *http://github.com/jpsthecelt/plutoraCreateSys.git.*

……

