Levi Lebhart

Geography 375 Python Programming for ArcGis

Nate Jennings

May 13th, 2018

Technical Aspects of Python Script

Purpose:

Ability to login into ArcGis online and download data automatically by using a configuration .ini, or through a tool on ArcGis online.

The Basic Process:

The script is first prompted by the user to input their login information into either the tool parameters or to have them input through a configuration file that is stored elsewhere the datapath is placed in the python script to read. The user is also asked to input the feature name that they wish to download from the online service so that the file can be searched for within Arc REST back end. The user is also required to input a file path that will be used to place the final .zip of the data which is then extracted.

How this is Achieved:

In order to get to the data that is stored behind a secure login page, the user needs to be able to generate a token in order to access it. A token is a small piece of information that authenticates a user's

access to data without the need to continually log in. It can be seen present in the ArcREST url, positioned at the very end as a combination of randomly assorted characters.

ArcREST is a simple directory used for server side interaction. The ArcREST urls have another version called “JSON”. JSON operates as a way to translate server features into something more readable for programming. The JSON within the rest URLs contain all of the information needed to help locate and pull out the information you need when querying for server side information.

The main hurdle with this script was trying to figure out how to get it to crawl it's way around the website in order to find the service item ID. The service item ID was pivotal, in that it's a unique identifier used when downloading the data off the server. Without the unique identifier there is always a chance of grabbing the wrong data.

The parameter variables are what were used to make sure that the write data was selected. Querying against a certain selection of variables grabbed from the JSON on the rest site allowed me to look for exactly what I needed and kick back what ever was unnecessary.

Once the proper identifier was received, the script then uses this information and places it into the download feature service definition. It logs into arc gis using the predefined user information and downloads the information into the predefined save path. This save path is dictated by the end user. Sometimes the file will already exist and then it will delete it and download the new one in it's place, and extract it as a file.

The next portion of the code is a set up page for the configuration file, it maps out the content of the .ini for python to look through, and plugs them into the appropriate variables found at the beginning of the script.

<u>What's it uses to do this:</u>

The main modules imported where os, sys, traceback, for basic functions and trouble shooting.

Tempfile was imported to be able to find the JSON values and hold onto them while trying to locate the specific file to be downloaded. Zipfile and configparser were also loaded into the modules in order to be able to set up a configuration file for the script, and allow the zip and unzipping functions within the downloading functions. The JSON was used so the script could read and interpret the information it was looking for on the website. The UUID module was imported so the script could be used from different computers while still remaining unique in their own identifiers and the data they access. The urllib and urllib2 modules were used to look at and open the needed URLs so the script could travel around and find the information it needed hiding within the JSON. Getpass was a module used to retrieve the user's password within the python environment. Arcpy, ArcREST and ArcGis modules used for ArcGis functions.

Challenges:

Trying to understand the REST and JSON ideas when it came to coding around scouring a server was a bit of a hassle, but once I realized the main ideas behind it, I was able to get it to work and things started to click into place a lot faster than expected.

I couldn't get the final formation to run at all and couldn't figure out how implement it as a tool process, so for the time being it's going to be something I need to come back too. But with some tweaking it should run as something for the

Primarily bit off more than I could chew even when trying to make it something that was as simple as possible, at least on paper. Programmatically however, is a completely different question.

Sources:

A lot of the formatting and styling was used from looking at the SyncSurvey script posted by

ESRI's James Tedrick. The modules he used were also tremendous in getting me into the right direction and showing me the tools necessary to get closer to completing what I was trying to do and will be referred too in the future. The “{0}&f=pjson&token={1}” was a really neat way he concatenated the url's within rest and was a big help in cleaning up the form of the code. I was also able to find help with trying to get the configparser set up by looking at code posted by mingrammer of github.

SyncSurvey.py-
https://community.esri.com/groups/survey123/blog/2016/12/14/migrating-data-from-the-survey-feature-service-to-an-enterprise-geodatabase?commentID=45053&et=blogs.comment.created

configparser:
https://gist.github.com/mingrammer/baa0242241ad86bf2ac174bc8b313ea6#file-using-external-file-config-ini-ini

https://gist.github.com/mingrammer/fe4440b750184ff658790ce0cb4d474c#file-using-external-file-py