Automated Feature Service Downloading

```
def downloadFeatureService(serviceItemID, username, password, outpath, featureService):
    arcpy.AddMessage('t\-Unique Service Item ID retrieved...')
    arcpy.AddMessage('t\-Begginging to download...')
    #uses the previous variables to establish connection to host and to begin downloading the feature service
    featureService = serviceItemID
    login = GIS(username, password)
    #uses id to direct straight to the file needed
    featureServiceDL = login.content.get(seviceItemID)
    save_path = (outpath + featureService)
    if arcpy.Exists(save_path):
        arcpy.Delete_management(save_path)

        featureServiceDL.download(save_path)
        #pulls it down as a zip file and then extracts it to the defined outpath
        zf = ZipFile(outpath + featureService + '')
        zf.extractall(path = outpath + featureService + '' + '')
```

```
#Modules
import os, sys, tracebac
import tempfile
import zipfile, configpa
import json
import uuid
import urllib, urllib2
import getpass
import arcpy, arcrest, arcgis

#Variables input by the enduser
username = arcpy.GetParameterAsText(0)
password = arcpy.GetParameterAsText(1)
featureServiceName = arcpy.GetParameterAsText(2)
outpath = arcpy.GetParameterAsText(3)
```

```
he ArcGis Online Cloud,
tps://www.arcgis.com'):

ad for logging in

ecessary
sed
```

```
    'username': username,
    'password': password,
    'client': 'referer',
    'referer': portal_URL,
    'expiration': 60,
    'f': 'pjson'
})
#input generates the token needed
tokenURL = '{0}/sharing/rest/generateToken?'.format(portal_URL)
response = urllib.urlopen(tokenURL, parameters).read()
#puts together the parameters and reads them to grab the unique token
token = json.loads(response)['token']
return token
```

Levi Lebhart
Geography 375- Python Programming for GIS
Prof. Nate Jennings
5/13/18

Modules Used and Variable Parameters Set

```
#Modules
import os, sys, traceback
import tempfile
import zipfile, configparser
import json
import uuid
import urllib, urllib2
import getpass
import arcpy, arcrest, arcgis

#Variables input by the enduser
username = arcpy.GetParameterAsText(0)
password = arcpy.GetParameterAsText(1)
featureServiceName = arcpy.GetParameterAsText(2)
outpath = arcpy.GetParameterAsText(3)
```

Begin to import the needed modules within the process:

- ArcRest, urllib, urllib2 and JSON will be pivotal for web processes
- UUID will be needed for unique ids so other machines can run it
- Os, sys, traceback, getpass, tempfile, configparser and zipfile will be used for local processes

Variables/Parameters are set up:

- Username, password and feature service name, and outpath could be set by a user through a custom scripting tool
- These parameters could also be applied through the config parser operation

Get Token Operation

```
#Operation to generate a secure token for session on the ArcGis Online Cloud,
def getToken(username, password=None, portal_URL = 'https://www.arcgis.com'):
    arcpy.AddMessage('\t-Getting Token...')
    #sets up the password and JSON parameters to be read for logging in
    if password == None:
        password = getpass.getpass()#prompts user if necessary
    #section describes what part of the JSON will be used
    parameters = urllib.urlencode({
        'username': username,
        'password': password,
        'client': 'referer',
        'referer': portal_URL,
        'expiration': 60,
        'f': 'pjson'
    })
    #input generates the token needed
    tokenURL = '{0}/sharing/rest/generateToken?'.format(portal_URL)
    response = urllib.urlopen(tokenURL, parameters).read()
    #puts together the parameters and reads them to grab the unique token
    token = json.loads(response)['token']
    return token
```

Start of the Get Token Operation:

- Sets the variables it will be using through out the process
- Starts by setting up parameters for the JSON to input the user's login information to be able to log into ArcGis online
- Logs the user in and retrieves a token for the session, giving the script access to search through out the REST server
- Source code taken from SyncSurvey.py from James Tedrick

Get Feature Service URL Operation

```
#gets service definition to allow re-uploading of the file
def getFeatureServiceUrl(token, featureserviceURL):
    arcpy.AddMessage('t\-Login successful.')
    arcpy.AddMessage('t\-Retrieving service url...')
    #looks for the url within the JSON object
    featureserviceQueryParameters = urllibe.urlencode({
        "url" : ""
    })
    #reformats the JSON object into a simple url
    serviceURL = ''.format(featureserviceQueryParameters)
    #the url is then re-attached to the other end of the URL
    grabUniqueUrl = '{0}&f=pjson&token={1}'.format(serviceURL, token)
    #gives the final url needed to get to the ID
    featureserviceURL = json.loads(grabUniqueUrl)
    return featureserviceURL
```

Get Feature Service URL:

- In order to access the required information to download the data, there needs to be a service Item ID, which is found within the JSON
- Sets JSON query parameters to search for the url field in the REST directory and then proceeds to copy it
- After copying the url out of the JSON, it then takes the token it had previously and combines that onto the end of the service url, allowing for access to the server's site
- “{0}&f=pjson&token={1}” concatenation taken from James Tedrick's SyncSurvey.py script

Search for Global ID Operation

```
def searchForGlobalId(featureserviceURL, token, featureFind):
    arcpy.AddMessage('t\-Service definition retreaved successfully...')
    arcpy.AddMessage('t\-Searching for unique item ID...')
    #pulls on the feature url and the token to show the JSON to pull the Global ID value out
    featureParameters = urllib.urlencode({
        "serviceItemId" : ""
    }),
    #opens up the URL and reads the ID and loads it into a temp
    featureFind = urllib2.urlopen(featureserviceURL, token, featureParameters).read()
    serviceItemID = json.loads(featureFind)['serviceItemID']
    return serviceItemID
```

Get Service Item ID:

- The feature on ArcGis online has a specific ID tied to it uniquely
- This allows us to download this specific piece of data and not accidentally grab another by mistake
- To achieve this, we first set the parameters for the JSON that we are interested in
- The main one is the “serviceItemId”
- We take the service url and run the parameters against it to retrieve the ID
- Once the ID is retrieved, we load it temporarily to be used as a variable in the next step of the proccess

Download Feature Service Operation

```
def downloadFeatureService(serviceItemID, username, password, outpath, featureService):
    arcpy.Add.Message('t\-Unique Service Item ID retrieved...')
    arcpy.Add.Message('t\-Begginging to download...')
    #uses the previous variables to establish connection to host and to begin downloading the feature service
    featureService = serviceItemID
    login = GIS(username, password)
    #uses id to direct straight to the file needed
    featureServiceDL = login.content.get(seviceItemID)
    save_path = (outpath + featureService)
    if arcpy.Exists(save_path):
        arcpy.Delete_management(save_path)

        featureServiceDL.download(save_path)
        #pulls it down as a zip file and then extracts it to the defined outpath
        zf = ZipFile(outpath + featureService + '')
        zf.extractall(path = outpath + featureService + '' + '')
```

Downloading the Feature Service:

- Simple proccess, uses the arcgis module to jump onto ESRI and use the end user provided credentials to login
- Once logged in it then runs a download function based on the service Item ID variable we just gathered in the previous step
- This data is downloaded as a zipfile, only after we've checked to see whether or not the file had already existed, if it already did, then it is deleted
- The file is then saved to the outpath defined by the end user through the script tool or the config file

Configuration File Set Up Operation

```
#optional set up to use with a config.ini for automated downloads of a certain service
def Config(config, configmap, configfile):
    configfile = '\\automationUserConfig.ini'#userdefinedpath
    #sets up map to read the configuration file
    [USER_INFO]
    portal = http//www.arcgis.com
    username = USER_1
    password = PASS_1
    service_url = http//www.arcgis.com
    sde_conn = '\\connection.sde'#userdefinedpath
    config = ConfigParser.ConfigParser()
    #reads and uses the information from the file if available
    config.read(configfile)
    portal = config['USER_INFO']['portal']
    username = config['USER_INFO']['username']
    password = config['USER_INFO']['password']
    service_url = config['USER_INFO']['service_url']
    sde_conn = config['USER_INFO']['sde_conn']
```

Configuration set up:

- First we set the file path where the configuration file is supposed to be located
- Once the file is found, the code maps out where the objects and keys are within it
- These objects are then configured to correspond with the variable we established at the beginning of the script
- This method of configuration for the config file was originally done by a github user with the handle of 'mingrammer', I simply modified it to make it work within the means I needed it too

Except Block for Trouble Shooting

```
except:

    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n     " +        str(sys.exc_type) + ": " + str(sys.e
    msgs = "ARCPY ERRORS:\n" + arcpy.GetMessages(2) + "\n"

    arcpy.AddError(msgs)
    arcpy.AddError(pymsg)

    print msgs
    print pymsg

    arcpy.AddMessage(arcpy.GetMessages(1))
    print arcpy.GetMessages(1)
```

Except block:

- Used to trouble shoot any issues there may have been with the script
- The one provided to us by Prof.Nate Jennings

Sources:

James Tedrick, ESRI:
https://community.esri.com/groups/survey123/blog/2016/12/14/migrating-data-from-the-survey

Mingrammer:
https://gist.github.com/mingrammer/baa0242241ad86bf2ac174bc8b313ea6#file-using-externa

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

Prof. Nate Jennings:

Https://pythonprimer.urbandalespatial.com/resources

ArcGIS for developers:
Understanding certain functions and Module usage