ART Student Loader Utilities

Instruction Manual

Description

The ART student loader utilities are a set of Python scripts designed to automatically load the millions of students from the nightly CALPADS dump file into ART's REST API. The utilities consist of three executable scripts and two settings files:

csv_downloader.py	This executable python script downloads the nightly dump file from the CALPADS sFTP server. Can auto-resume an aborted download where it left off. Will automatically download this morning's dump file on the server using the existing naming convention, and will extract the raw CSV data from the zip format used. Run with -h or --help for a usage summary.
csv_load_students.py	This executable python script reads the student data from the CSV file downloaded by csv_downloader and loads it into ART via the ART REST API. The utility can be instructed to start at any offset in the data file in case a previous run was aborted, and can also be given a number of students to upload instead of uploading the entire file. Run with -h or --help for a usage summary.
launcher.py	This executable python script displays a GUI that can automate both scripts above. This allows the user to make various settings to the values used by both scripts before clicking 'GO'. This will start both the downloader and uploader in separate output boxes so the user can watch them both run in parallel. It's mainly designed for one-off execution, testing, and troubleshooting. The utilities are primarily designed to be run unattended by cron on a daily schedule.
settings_default.py	This file is distributed with the utilities and contain default settings used by all the tools. It's mainly designed to be a template you will copy to settings_secret.py before modifying. If settings_secret.py is missing this file will be read, but a warning will be printed as you probably won't connect to the right servers.
settings_secret.py	This file is NOT distributed with the utilities. It's a settings override file you

	will create and modify, and hopefully prevent others from reading via tight permissions, etc. This file contains all the sensitive passwords and URL's, etc, the utilities read to connect to ART and the sFTP server where the sensitive data is stored. Copy settings_default.py to this file and modify to taste.

Script Execution

csv_downloader.py

```
Downloads today's student CSV dump from CALPADS sFTP server.
Please put your settings in settings_secret.py. Defaults are in settings_default.py.
Help/usage details:
  -f, --filename           : local filename to write into
  -r, --remotepath         : remote filepath to download (example: './Students/CA_students_20171005.zip')
  -o, --offset             : where to start reading / writing in the files, in bytes
                             (will resume at end of any pre-existing file or at beginning of a new file)
  -h, --help               : this help screen

The script is designed to be run with no arguments, with all settings coming from settings_secret.py:
$ ./csv_downloader.py
Or to grab a one-off file from another directory:
$ ./csv_downloader.py -f my_local_file.csv -r ./Requests/special_dump.csv
```

Usage notes

If no offset is provided, the script will try to be smart:

- If no local file is found, it will download the whole file (offset 0).
- If a local file is found and it's smaller than the remote file, it will RESUME the download.
- If the remote file is smaller than the local file, it will print an error saying so and exit.
- If the remote file is the same size as the local file, it think it's done and will exit.

If you provide an offset that is smaller than the local file, the script will truncate the local file at that offset and then resume downloading it at that spot.
It will never automatically truncate the local file, but it might append the wrong file contents to the end of an old local file if the remote file is larger (it thinks it's auto-resuming). So use caution with pre-existing local files - the script does not check the local file contents against the server.
Generally when in doubt it's safer to just delete the local dump file and redownload the whole thing.

csv_load_students.py

```
Loads students from CSV file into ART via REST API.
Please put your settings in settings_secret.py. Defaults are in settings_default.py.
Help/usage details:
  -y, --dryrun             : do a dry run - does everything but actually POST to ART
  -f, --file               : file to read students from
  -e, --encoding           : encoding to use when reading file
  -d, --delimiter          : delimiter to use for CSV format
  -o, --offset             : where to start reading in the file, in bytes (defaults to byte 0)
  -n, --number             : the max number of students to upload
  -h, --help               : this help screen

The script is designed to be run with no arguments, with all settings coming from settings_secret.py:
$ ./csv_load_students.py
Or to load 20 students from extra.csv starting at the one millionth byte in the file:
$ ./csv_load_students.py -f extra.csv -n 20 -o 1000000
```

Usage Notes

When in doubt, it's generally a good idea to do a --dryrun before uploading for real. Nothing will be uploaded to ART, but the file will be read and authentication will be performed.

Installation and setup

The utilities require Python 3.4 or better. They've been tested against versions 3.4 and 3.6 (3.6 is recommended). No other Python versions are supported. The package requirements are in the requirements.txt file (currently 'requests' and 'paramiko' are the only libraries needed). Tkinter is also needed to run the GUI in launcher.py. Installation is detailed below for Ubuntu and CentOS. The scripts have also been tested on MacOS Sierra with Python 3.6.2 installed via brew. It should run fine on Windows and other platforms if Python 3.6 and all the requirements are properly set up.
Instructions are included below for setting up your Python 3 environment on Ubuntu and CentOS.

Ubuntu 14.04

```
Become root or use sudo for all '#' commands, be regular user for '$' commands.
-- base setup
# apt update && apt upgrade
-- set up python 3
# apt install python3
# apt install python3-pip
# apt install build-essential libssl-dev libffi-dev python3-dev  # for paramiko/ssl
# apt install python3-tk  # only if you want to run the GUI in launcher.py
-- set up virtual environment for an isolated python
# apt-get install python3.4-venv
$ cd <loader-script-directory>
$ mkdir environments
$ python3.4 -m venv art34
$ source environments/art34/bin/activate  # enter the art34 env. should show a nice env prompt.
-- now all regular python and pip commands will use your art python 3.4 environment
$ python -V  # show python version. can also run pip -V to see that pip is OK
$ pip install -r requirements.txt

You're all set! Make sure to always enter the correct python environment before running the loader scripts,
or you may start up the wrong python version or be missing packages.
```

CentOS 6.9

```
-- become root or use sudo for all '#' commands, regular user for '$'
-- base setup
# yum -y update
# yum -y install yum-utils
# yum -y groupinstall development
-- set up python 3, using ius packages --
# yum -y install https://centos6.iuscommunity.org/ius-release.rpm
# yum -y install python36u
# yum -y install python36u-pip
# yum -y install python36u-devel
# yum install -y python36u-tkinter  # only if you want to run the GUI in launcher.py
-- set up virtual environment for an isolated python
$ cd <loader-script-directory>
$ mkdir environments
$ python3.6 -m venv art36  # can change art36 to whatever you like
$ source environments/art36/bin/activate  # enter the art36 env. should show a nice env prompt.
```

```
-- now all regular python and pip commands will use your art python 3.6 environment
$ python -V  # show python version. can also run pip -V to see that pip is OK
$ pip install -r requirements.txt  # run from within the loader script directory
```

You're all set! Make sure to always enter the correct python environment before running the loader scripts, or you may start up the wrong python version or be missing packages.

If you prefer epel or Python 3.4, you can adjust the commands like the following (changing all 36u to 34):

```
# yum install -y epel-release
# yum install -y python34-tkinter
... etc.
```

Configuration

Settings, overrides, and security

The scripts are configured by reading settings files. This way you can run the scripts with no arguments and they'll do what you want every time.
settings_default.py is distributed with the scripts in github and contains sensible, but public default settings. To provide your own values, **copy settings_default.py to settings_secret.py** and adjust as desired. The scripts look for settings_secret.py first and will only load the defaults if the secrets are missing (it will also complain if it has to do this, as that's probably not what you want).
If you put any sensitive values in settings_secret.py it's advised to **set the permissions on settings_secret.py so nobody but you can read it**. It also shouldn't be put into source control or emailed around.

Authentication

Downloader sFTP authentication

Depending on how the sFTP site you're loading from is configured, you may need a private key file and potentially a password to unlock that key file, or the sFTP site may only require a username and password. All of these settings are configured in your settings_secret.py file as described above. The values look like this:

```
SFTP_HOSTNAME = "8.8.8.8"  # Your sFTP hostname
SFTP_USER = "ubuntu"       # Your sFTP username
SFTP_PASSWORD = "userpass" # Your sFTP user password, if using password authentication
SFTP_KEYFILE = "/Users/ubuntu/.ssh/art-capacity-test.pem"    # Your sFTP user's ssh private key file
SFTP_KEYPASS = "keypass"   # A password to unlock SFTP_KEYFILE if encrypted, else None if not needed
```

Downloader key format

The key used for sFTP by the downloader is just a regular ssh key. It should be an RSA private key in PEM format. This same file can be used directly by the ssh command on Linux and OSX. In fact, putting this key in your ~/.ssh folder and trying to log in to the sFTP server with ssh is a great way to test the key before trying it with the script.
Note: A putty key (.ppk) can be converted to the correct format with puttygen. Puttygen comes with PuTTY and is available for Windows and MacOS (via brew). On MacOS this command makes a supported sftp_priv.pem file:

```
$ puttygen SomePrivatePuttyPPK.txt -O private-openssh -o sftp_priv.pem
```

Troubleshooting

Runtime issues

Rabbit can get stuck when loading 6M students. I saw it fail with 2.4M messages in the 'student' queue when the watermark was set to 0.4 on art-capacity-test (those are big boxes with 60GB RAM). If you see a queue in 'flow' state, it's stuck. It caused the loader script to be refused a connection, which aborts the script.
On capacity-test the watermark was set to 0.4 (40% of memory; about 24GB). You can tweak this if your machines have enough RAM by setting the watermark higher, for example 99%, as follows. Of course, if you set this too high it can cause other issues as erlang will suck up all your RAM:
(you maybe also put the setting in the config file ot make it permanent)
Show limits:

```
$ sudo rabbitmqctl eval 'vm_memory_monitor:get_vm_memory_high_watermark().'
$ sudo rabbitmqctl eval 'vm_memory_monitor:get_memory_limit().'
```

Set limit (can do ratio or absolute - I used ratio):

```
$ sudo rabbitmqctl set_vm_memory_high_watermark 0.999999
$ sudo rabbitmqctl set_vm_memory_high_watermark absolute 40000MB
```

3.2M students in queue caused erlang to show over 17 GB RSS RAM on art-capacity-test.
0.75 is probably a more reasonable number - could probably use 0.6, but cuts it close for 3.2M students.

Performance

The scripts running on a CentOS host in us-east-1 downloaded the csv from a us-west-2c box at about 3 megabytes/sec (slow) and loaded into ART at about 2100 students/sec (that's slow). Running the scripts from the same availability zone is much faster. Fastest yet is to run it on the same box as ART.
The downloader was faster than the loader in all tests.
Remember that uploading only puts the students into rabbitmq, on the 'student' queue. ART then process each student, putting each into mongodb after processing. The latest ART takes less than an hour to drain over 6M duplicate students from the student queue because we recently added code to skip dupes.
If all 6M students are 'new', ART will take many hours to drain the queues, because it reads each student from the student queue, saves it to mongo, then puts it on the events.in queue. ART will then load each student from events.in and run implicit eligibility rules on it, which were recently optimized but are still quite involved. You can see what's in the rabbitmq queues by pointing a browser at your art host, port 15672 (by default).