**Documentation**

The intention of this software for me is solely for deleting the Fortnite video files from my PC after they have successfully been uploaded to the Synology DiskStation. As NVIDIA GEFORCE NOW saves the gameplay captures on my local hard drive as I'm playing the game, which causes low disk space on my (C:) Drive, therefore after uploading these video files to my Synology DiskStation using the app called Synology Drive, this software would delete those local video files. I'll try my best not to hard code this software, so it can be of more general use.

**Author:** Danish Naseem

**Name:** EyeWatch

**Version:** 1.0

**Date Started:** 01/16/2020

**Language(s) Used:** Java, HTML5

**Dependencies:** JDK 13.0.1, JUnit5.4, JSONObject, apache-httpcomponents-client-4.5.11

**API(s) Used:** Synology FileStation

**Database Used:** CSV (Comma Separated Values)

**Data-Structure(s) Used:** HashMap (for reading keys and values out of the .csv file), and LinkedList (for storing the filenames)

**Lines of Code:** 1343

**Copyright:** 2020 © Danish Naseem

**License:** MIT

**-- INSTRUCTIONS --**

- Refer to page 26 of the Synology File Station API Guide for the `list` method to enumerate all the files within the specified folder path. Then compare all the filenames with the selected local filenames. And if they exist then and only then Delete the local files from the Windows PC.

**-- WHAT THE PROGRAM DOES --**

Scans the selected directory, and if it contains any files, add those files' filenames to the ‘toSearch’ LinkedList and searches for those filenames inside the User Selected Synology NAS directory. If any of those filenames match, add their filenames to ‘toDelete’ LinkedList. After the search process is over. Delete the file(s) from the local directory, whose filenames matches the ones in the ‘toDelete’ LinkedList

**-- DISCLAIMER --**

- This program has only been designed to work with the Synology API. Using it with any other remote server or API would end up producing unknown errors.

- This program ONLY searches for files inside the selected remote directory and selected local directory, and NOT its sub-directories.

- This program is only intended for Windows and has been tested on Windows 10.

- Only HTTPS port number will be accepted.

**-- TODO --**

- Call the SynologyAPI’s run method from the Lib.java constructor and save the Filenames in a LinkedList.

- Change the name of the entire program from EyeWatch to EyePurgeWatch.

- Make a method in Lib.java that compares two Linked Lists (in this case they’ll be the remote files linked list and local files linked). This will have O(n^2) complexity, as it will compare each item from each linked list, hence running in a nested loop. While running in a loop, it will call the delete file method for the item that is present in both the linked lists.

- Fix either GUI.java or OperatingSystem.java so that the database doesn’t save setting until the monitoring has started correctly.

- Fix the port number error in the SynologyAPI.java

- Figure out the bug that is causing the GUI to not display the https port number error if the typed https port number is not an integer.

- Make the WatchLocalDirectoryStateThread class in OperatingSystem.java more efficient so it doesn’t take up the CPU when running in Idle. Create idle method to achieve this.

- Write the constructor for Lib.java to connect to everything and finally make the entire program run.

**-- LEGEND --**

= Synology API GET requests.

= GUI window options.

= Methods of different Java Classes.

= CMD commands.

**-- HOW THE PROGRAM RUNS --**

- User clicks on program icon and the program starts launching. It will look for the user settings from an already created Database, or .csv file (not sure what to use yet). If none is present, then a new one will be created and written with the variable names as keys and empty strings as values. Then the GUI will load up (***\*\*refer to GUI section under CLASSES for more details\*\*)*** , and by default will already have the information entered from the Database or .csv file (which right now is just empty strings, so GUI TextFields would be empty).

- Then user is going to enter the information and click on `Save Settings` button. After which the entered information will be recorded inside the Database or .csv file, replacing the empty strings with actual values.

- Then respected functions would be called to verify the information written inside the Database or the .csv file. If all is good, the program will run (***\*\* refer to `WHAT THE PROGRAM DOES` section above to see how it runs \*\*).***

- If something is wrong, the program will notify the user either by popup window or just by writing inside the log of the GUI. In which case, the user will have to re-enter the information and again click on `Save Settings`. Again, the information will the written to the Database or the .csv file, overwriting the existing values. Then again, the program will verify the information and if all is correct, the program will run, if not, this same continues.

- If the user clicks on ‘Browse’ button under `Local Directory` label, method `openLocalFileExplorer` would run

- If the user clicks on ‘Browse’ button under `Remote Directory` label, method `openRemoteFileExplorer` would run. This method first calls `checkRemoteFields` method, which checks if the Host name, Port number, Username, and Password fields are not empty. Then `openRemoteFileExplorer` calls `getHostname`, `getPortNumber` (this method verifies that the entered value is an integer), `getUsername`, and `getPassword`, and collects and stores theirs values in variables. Then with the gathered information, `openRemoteFileExplorer` calls `accessRemoteDirectory` inside SynologyAPI class. `accessRemoteDirectory` method calls `getAPI\_Info` (look at Step-1), `authenticate` (look at Step-2), and other various methods within SynologyAPI class. These methods do the following:

Step-1: Retrieve API information

Retrieve API information from the target Synology DiskStation by making a request to `[HOSTNAME]:[PORT\_NUMBER]/webapi/query.cgi` with `SYNO.API.Info` `API` parameters. The information provided in the response contains available API name, API method, API path and API version. Once you have all the information at hand, your application can make further requests to all available APIs, which here will be the FileStation API. Our request GET to get all the available APIs:

`https://[HOST\_NAME]:[HTTPS\_PORT\_NUMBER]/webapi/query.cgi?api=SYNO.API.Info&version=1&method=query&query=all`

And to be more precise, our GET request to get Auth and FileStation.List API info is:

`https://[HOST\_NAME]:[HTTPS\_PORT\_NUMBER]/webapi/query.cgi?api=SYNO.API.Info&version=1&method=query&query=SYNO.API.Auth,SYNO.FileStation.List`

If we successfully receive a .JSON file in response, we will extract SYNO.API.Auth and SYNO.FileStation.List dictionaries from the Data dictionary.

But if we receive an ERR\_CONNECTION\_TIMED\_OUT error, we will inform the user that hostname or port number is invalid.

Step-2: Login

Now we know FileStation is available, if not display an error. Now we begin the login process by simply making a request to the SYNO.API.Auth API using the provided username and password. The POST request is the following:

`https://[HOST\_NAME]:[HTTPS\_PORT\_NUMBER]/webapi/auth.cgi?api=SYNO.API.Auth&version=3&method=login&account=[USERNAME]&passwd=[PASSWORD]&session=FileStation&format=cookie&otp\_code=[TWO-WAY-AUTHENTICATION-CODE]`

TWO-WAY-AUTHENTICATION-CODE is a one-time pin code generally obtained from a mobile phone.

If we receive {“error”: 400} and {“success”: false} inside the JSON file, we display an incorrect username of password error to the user.

If {“success”: true} then we get an “sid”, which we’ll save in a variable to use in making other API requests. But since out `format=cookie` in the above GET request, so we don’t have to use “sid” in making API requests. As “sid” will saved as a cookie named `id`

Step-3: Request a File Station API

- After Everything has gone right and the user has successfully logged in to the Synology DiskStation’s FileStation. The program saves the local directory path and remote directory path inside our custom .JSON file for the program to use. The GET request is as follows:

`https://[HOST\_NAME]:[HTTPS\_PORT\_NUMBER]/webapi/entry.cgi?api=SYNO.FileStation.List&version=1&method=list&additional=real\_path%2Csize%2Cperm%2Ctype&folder\_path=%2F[FOLDER\_PATH]`

For me the FOLDER\_PATH is = `home/Drive/Videos/Other/NVIDIA/GeForce%20NOW/Fortnite`

Step-4: Logout

- The program logs out of the Synology drive and starts running in the background keeping track of any changes inside the local directory. If any changes, then the program logs back into the Synology DiskStation’s FileStation and runs the process. Once the process finishes, it logs back out. The GET request for logout is as follows:

`https://[HOST\_NAME]:[HTTPS\_PORT\_NUMBER]/webapi/auth.cgi?api=SYNO.API.Auth&version=1&method=logout&session=FileStation`

- If the local directory is empty, it does nothing. If any files are added to it, the program runs. ***\*\* refer to `WHAT THE PROGRAM DOES` section above to see how it runs \*\****

**-- MANAGING THE DATABSE (.CSV FILE HERE) --**

- We will create a directory called `.EyeWatch` in the program directory.

- Inside will create a file called `SaveSetting.csv`.

- This file will be initially with keys and their empty values (‘\*’)

- This file will oversee saving the user setting after the user has clicked on the `Save Setting` button.

- Then the file will be overwritten with the values that the user has entered.

- The program will read from this file, gather the keys and values in a HashMap, which later will be iterated and each key and its value will be saved in a variable, for the program to further process and pass those variables to other method as arguments.

- Since the `.EyeWatch` should be hidden and not be seen or managed by the user, it will be hidden.

- The following CMD command will be executed to do.

TO HIDE THE DIRECTORY:

**Command:** `attrib +s +h "[PATH/TO/THE/DIRECTORY]"`

TO UNHIDE A DIRECTORY:

**Command:** `attrib +s -h "[PATH/TO/THE/DIRECTORY]"`

**-- CLASSES: [GUI, GUIHandler, SynologyAPI, Main, Testing] --**

1. GUI [`getHostname`, `getPortNumber`, `getUsername`, `getPassword`]

*Rough Sketch*

*Current GUI About Window Design*

**

*Current GUI Design*

**

*GUI Design Description*

First Row

* description

Second Row

* empty space

Third Row

* `Local Directory` label

Fourth Row

* Column 1
  + input panel for either user to enter the path to the local directory or
* Column 2
* by clicking on the `Browse` button right next to the panel and choose the local directory using the file explorer

Fifth Row

* `Remote Directory` label

Sixth Row

* Column 1
  + `Host name or IP address` label
* Column 2
  + input panel for above label
* Column 3
  + empty space
* Column 4
  + `Port Number` label
* Column 5
  + input panel for above label

Seventh Row

* Column 1
  + `Username` label
* Column 2
  + input panel for above label
* Column 3
  + empty space
* Column 4
  + `Password` label
* Column 5
  + input panel for above label

Eight Row

* Column 1
  + `2 Step Verification Code` label
* Column 2
  + input panel for above label

Ninth Row

* Column 1
  + input panel for user to enter the path to the remote directory

Tenth Row

* Column 1
  + `Run on Startup` checkbox.

Eleventh Row

* Button saying 'Done’

1. GUIHandler [`openLocalFileExplorer`, `checkRemoteFields`, `openRemoteFileExplorer`]
2. SynologyAPI [`accessRemoteDirectory`, `getAPI\_Info`, `authenticate`]
3. Lib
4. Testing
5. Main
6. OperatingSystem[`runOnStartup`, `runInBackground`, `watchLocalDirectoryState`]

If the user checks the `Run on Startup` checkbox, then we add the program to the Windows 10 Startup folder, whose path is as follows: `C:\Users\[USER]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`

To get the current [USER], make a call to CMD with this command: `echo %username%`

This command would give the username in the form of a String