

Data Pull Automation Proposal

Automation

David Healy, Jeff Underhill, Kathleen Errington, Marcus Chase and Zachary Thoroughgood | NHDES PA Monitoring Project | 07/12/2022

# Introduction

We are tasked as a team to tap a new resource that is up and coming for data pulling not only in New Hampshire; but also, other states and all over the world. This technology being a monitor that is so small it can fit in your hand but is seemingly inexpensive when put up against a full-fledged data collecting field station. If we can correctly pull data and collect ten-minute averages; we can expand the data collection across the state at a cheaper cost and, it is a great way to interact with the public in a collaboration of resources for the greater good. So how far have we come so far?

* With the help from our good friends at Keene State College (JC Woodward and Dr. Traviss); we took in their current model for pulling data from the Purple Air Monitors through their two python scripts of (Fetch.py and MapMaker.py) and formulated a plan for our own model.
* Currently, we have a program called Pull.py that can pull instantaneously the data given back from a call to the Purple Air API servers for every unit that is listed in the configuration file (an .ini file) and reports all the data back through a .csv file that is stored. The program also logs everything during execution; every execution with a time stamp to help keep track of any errors. This program can also recognize whether we have a .csv file created for the day or not to either append to the existing or create a new one for the beginning of the day.

## Where we’re at?

We are at the final stages of our first deliverable, which is a working automation program that will be set up to run for our first testing period inside of our contained testing environment. This program would be tested to run for an overnight at first and if all goes accordingly, it will go for a week straight of pulling to analyze the effectiveness of the program to debug and improve it to continue our steps in pulling the data from the Purple Air Monitors effectively/efficiently.

## What we need Assistance With?

Well, we have realized that the model provided by JC and Dr. Traviss was continuously running their pulling code until a breakage occurred inside a *while True* loop; which was verified by JC via email:

“*You are correct on every point.  That is the main loop in the program that makes a map then sleeps for ten minutes until it's time to make another one.  There is an obsolete PC not usable for much else that is dedicated to running this one program,”* (Email from JC Woodward on 7/11/2022 at 5:37 P.M.)

Given this verified information there are a few routes we could take as alternative matrices for automating our Pull.py.

# Automation Details

Now we will investigate a few ideas/alternatives to automating the Pull.py to pull our much-needed data from the Purple Air Monitors.

## Alternative Matrices

* Purchase a raspberry pi which is inexpensive at $50 or less from Amazon.com or [Buy a Raspberry Pi – Raspberry Pi](https://www.raspberrypi.com/products/).

Pros:

* Cheap device/affordable
* Continuously run the program without fear of damaging device
* It is small and can fit in any environment

Cons:

* Does not have a lot of storage
* Linux environment which means we would have a while True loop as well
* If it goes down; it would need a manual restart of the program
* Task Scheduler on the Modeling Computer

Pros:

* We already have both
* Task Scheduler can wake up the computer to run automation of a program
* The computer can also be put back to sleep using Task Scheduler
* Modeling computer has lots of storage
* We will have an abundance of control as to when the program runs and how frequently
* Eliminates a lot of human error
* Minimal Servicing

Cons:

* We will need to consider unexpected events such as power outages
* The computer may never sleep unless an unexpected event occurs
* Continuous While Loop on the Modeling Computer

Pros:

* Eliminates a lot of human error
* Continuously grabs our data

Cons:

* Life expectancy of the computer severely decreases
* Puts it at risk for unexpected events
* Would need a manual restart in unexpected events
* Possible loss of data when no one is here to service it
* Hybrid of Task Scheduler on the Modeling Computer

Pros:

* Gives the modeling computer a business days rest
* Allows for a seemingly errorless conversion of pulling
* Still eliminates a lot of human error
* Allows for a hybrid of both full automation and manual conversion
* Already have the resources

Cons:

* Human error (forgetting)
* Holidays/Weekends (However the program will run automatically still with breaks through going down to sleep mode until being awoken again)
* Minimal service from overlapping and taking over the ongoing shift

## Best Choice of Solution

Arguably, the best choice would be the hybrid which would consist of allowing the modeling computer to run automatically overnight every ten minutes by being awoken from sleep, and then being put back to sleep until the next morning when a team member can time the overlap of execution. Meaning once it runs, they have a ten-minute window to toggle disabling the task scheduler on the modeling computer and shutting it down to give it some rest and enabling the preloaded task scheduled automations (which would be exact duplicates as the modeling machine) to run for the workday on their own desktop. Thus, giving the modeling computer a chance to power down and reboot itself as well as putting the whole computer to rest in order to get the most out of its life expectancy.

This also means that the automation continues while the team member continues to work as ordinary throughout the day. The computer may have a pop-up command prompt every ten minutes for a couple seconds; but otherwise, it would be a great way to give the modeling computer rest and continue the data pulls.

# Steps for Automating S: Drive (Shared Version)

## Prior Requirements to setup

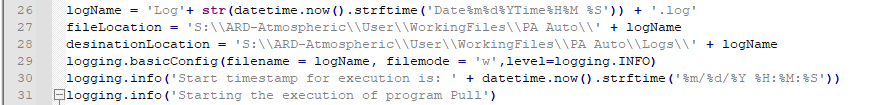
The testing location needs to first be set up to mimic this screenshot:

### Image 1-A

Graphical user interface, text, application

Description automatically generated

### Image 1-B



At the variable ***fileLocation***and ***desinationLocation***you will need to change the folder location addresses; that way it throws the log files inside your appropriate log file folder or else it will not know where to put those/look for those which will cause an error in execution that is fatal at this moment. You can find that location here in the command prompt:

### Image 1-C

Text

Description automatically generated

Only change the parts within ‘ ‘ and don’t get rid of logName or else you will have a bad time. Also, when you copy and paste that address over make sure to take out the > at the end and every forward slash needs to be two like my example ***Reference Image 1-B line 27-28.*** The ***desinationLocation*** needs to be the address of the folder and not the program which in the case of my example is just one step further into the log folder. You will also need to do a similar process for creating your batch file for automation:

### Image 1-D

Graphical user interface, text, application, Word

Description automatically generated

This batch file needs to include the > and is the address of the program “Pull.py” and cd is just saying ‘we need to access this address’. Once you change the address to be the same as the location your python file is in; you are good. The rest can stay the same unless you named Pull.py something different like Push.py. This batch file will also be saved in the same place as your python pulling program ***Reference Image 1-A.***

1. Place the Pull.py into a S: drive folder that everyone of the team members can access and name it.

### Image 1-E

A screenshot of a computer

Description automatically generated with low confidence

# Steps for Automating C: Drive (Local Version)

## Prior Requirements to setup

Before we start on the setup, you will need to have your location of operation looking like mine somewhere within your user account in the C: drive. ***Reference Image 2-A.***

### Image 2-A

Graphical user interface, text

Description automatically generated

As shown in ***Reference Image 2-A.*** you will need to have a folder to store the program logs from execution and a folder for your .csv archives. The naming will not matter but you will need to change a few file locations to those names you set. Do not worry we will do this thing together so no sweat. You will obviously need the Pull.py program, the batch files of move and start, and the demo.ini template because it includes a header so long that if you try to rewrite that you will surely get a headache. I would also suggest that you create a series of conventional and appropriate folders as I did inside of ***Image 2-A*** as you see fit***.*** Also, it would be a good idea to have Notepad++ to edit some files coming up!Once your file location is populated just like ***Image 2-A,*** and you have Notepad++ downloaded;let us continue to Step 1 below.

## Step 1 (Explanation of Demo.ini)

Let us first look at the *Demo.ini* file within your location of operation. Double clicking this file should open it up as a *.txt* file within a newly created notepad. If not, do not fret; right click the file and hit open with and then notepad. Now that we are there, you should see a file like this: ***Reference Image 2-B.***

### Graphical user interface, application Description automatically generatedImage 2-B

The value of api will always remain true; no matter what. The value of units can be changed with the unique ID tags you were given for your units. If you put more than one unit in follow this syntax: \*SINGLE UNIT\* = 151708 \*MULTIPLE UNITS\* = 151708|143456. Basically, units should be separated with the | character with no spaces. This convention is very picky to follow the exact syntax as the *Demo.ini* template portrays and always check your spacing to make sure it is correct. Keywords is similar but instead of separating with a | you will need to separate each keyword with a comma (,). These keywords can be found either on the Purple Air website: [API - PurpleAir](https://api.purpleair.com/) under the sensor data fields section and it can also be found in the *.json* files that are created for each unit after the python program’s execution.

Finally, duplicates is just for duplicated keywords which you can specify whether you want that keyword within ‘stats’, ‘stats\_a’, or ‘stats\_b’ as shown: ***Reference Image 2-C.***

### Image 2-C

Text, letter

Description automatically generated

As seen in ***Image 2-B,*** we will take the keyword pm2.5 for example. It has three different values within the sections of ‘stats’, ‘stats\_a’, and ‘stats\_b’ as said prior. If you want the value of pm2.5 inside of ‘stats\_b’; you would just need to put pm2.5:4 inside of the duplicates section inside of the *.ini* file: ***Reference Image 2-D.***

### Image 2-D

Graphical user interface, text, application, Word

Description automatically generated

We will most likely not need that because we are pulling all the data anyway with our long header. That is just for later down the road when parameters are tightened up for the header. Lastly for step 1, if you do not have an .*ini* file and you start the Pull.py program; it is not the end of the world. It will create a default template of it, and you can put anything you would like inside of it just like I have explained in this step.

## Step 2 (Updating Move.bat)

Let us right click now on the *Move.bat* file; ***Reference Image 2-B*** and click on edit with Notepad++. This should now open the file up in Notepad++ with a similar view to mine: ***Reference Image 2-E.***

### Image 2-E

Graphical user interface, application

Description automatically generated

I want you to now open the command prompt from your search bar down by the start put. Type in cmd and hit enter for the black window to open like: ***Reference Image 2-F.***

### A screenshot of a computer Description automatically generated with medium confidenceImage 2-F

Now that we are inside of the command prompt you should be within your C: drive already. If you are not, simply type C: and hit enter. Now you are, which you’ll just need to get to the location you put our files inside of. By typing Dir you can see where you’re at within the directory and what’s inside: ***Reference Image 2-G.***

### Image 2-G

Text

Description automatically generated

This shows where you should be because as you can see inside: ***Reference Image 2-F,*** there are the files and folders that are within: ***Reference Image 2-B.*** Getting there is simple, just look at your file explorer and see where the file is in there and use the command word of cd to climb through to the correct location. It would look like this: ***Reference Image 2-H.***

### Image 2-H

Text

Description automatically generated

This is how you would step through to the correct file location and cd.. is also how you step back as show inside of: ***Reference Image 2-H.*** Now going back to the *Move.bat* file as seen inside of: ***Reference Image 2-E,*** we will need to now copy this address that we stepped into which contains our location of operation into the *Move.bat*: ***Reference Image 2-I.***

### Image 2-I

Text

Description automatically generated

The next part is simple and just involves copying the last part we put in. As long as you have this folder for ***TenMinArchives*** in the same place as everything else, we can just do this to save us time: ***Reference Image 2-J.***

### Image 2-J

Graphical user interface, application, Word

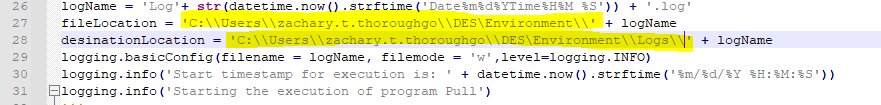
Description automatically generated

As seen inside of: ***Image 2-J,*** you just copy and paste the same address into the line 6 area and add a \ with the name of the folder. That is it, save this *Move.bat* file and it is onto the next step.

## Step 3 (Updating Pull.py)

To keep this short and sweet because the basis was covered inside of Step 2; you are going to be updating the file location of where the logs need to be stored which is inside of Pull.py which can be edited by opening with Notepad++. Then go to the lines of 26-31: ***Reference Image 2-K.***

### Image 2-K



As shown inside of ***Image 2-K,*** the variable of fileLocation is where you can just paste in the same address as previously put in the *Move.bat* before the folder was added. Make sure you do double back slashes, keep the address within the ‘ ’, and keep the tail of + logName. The variable of desinationLocation will be the same as fileLocation just with the Logs folder now added as show inside of: ***Image 2-K.*** Save this version of *Pull.py* and you are done this step and onto the next step.

## Step 4 (Updating Start.bat)

Another short and sweet version; you will need to right click and open *start.bat* with Notepad++. Then you are going to take the same address that we have been pasting in without the folders and putting it inside the file as shown: ***Reference Image 2-L.***

### Image 2-L

Graphical user interface, text, application

Description automatically generated

Once it looks like this, save the file and we are onto the next step.

### Revision (07/13/2022)

Your file will need to look like this due to the emergence of a new program I created with Visual Basic Script. This newly introduced script will allow for the previous *start.bat* to automate without creating a command prompt pop-up window, which is really distracting to the focus when you are trying to work whilst that is pulling. Now the updated version of this *start.bat* file will look like: ***Reference Revised-A.***

#### Revised-A

Graphical user interface, text, application, Word

Description automatically generated

Now as seen inside of ***Revised-A,*** you can see now we took off *@echo off* because we no longer need it. The cd (destination to a directory) remains unchanged and should be still suited with the same address as needed from the *Move.bat* file. That remains unchanged; however, the start call now has a peculiar *\min* in front of python now. Why might you ask? This starts the python program of Pull.py minimized; meaning you will never see the python window come up unless you are staring at the task bar. That is it for the revision; simple enough.

## Step 5 (Updating Starter.vbs) Revised (07/13/2022)

Now, to start the file should look something like this when opened using Notepad++ with a right click on the file: ***Reference Revised-B.***

### Revision (07/13/2022)

#### Revised-B

Graphical user interface, text, application

Description automatically generated

You will need to change the address to the same as the *move.bat* and *start.bat* because it will need to find the *start.bat*. This program essentially will run the

## Step 6 (Automation With Task Scheduler)

### The first thing we need to do is search Task Scheduler up in the search bar just like we did for command prompt in: ***Image 2-F,*** and click on it. You should now see a window like this: ***Reference Image 2-M.***

### Image 2-M

Graphical user interface, text, application

Description automatically generatedNow you will need to right click on the Task Scheduler Library tab which will bring up a menu like this: ***Reference Image 2-N.***

### Image 2-N

Graphical user interface, application

Description automatically generated

Now click Create Basic Task to begin the process of creating our automation. First you will need to name your task and you can give it a brief description of what it does. Then hit next as shown: ***Reference Image 2-O.***

### Image 2-O

Graphical user interface, text, application, email

Description automatically generated

Now you want to describe the trigger which for our needs will be daily so you can go ahead and click next if it looks like: ***Reference Image 2-P.***

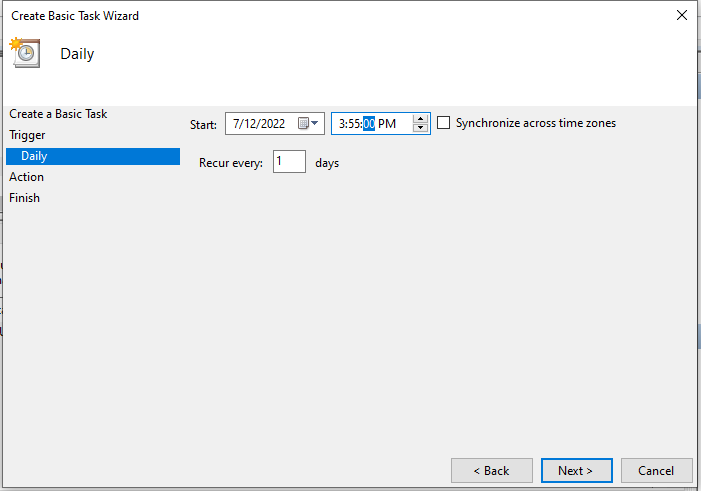
### Image 2-P

Graphical user interface, application

Description automatically generated

Now you’ll need to specify what day and time you want to start this task on just like: ***Reference Image 2-Q.***

### Image 2-Q



Once you have the credentials you want, you can continue to action by hitting next. Which will now look like: ***Reference Image 2-R.***

### Image 2-R

Graphical user interface, text, application

Description automatically generated

Which we want to start a program, so we leave it as default and click next to continue. Now you will need to click browse and find the *start.bat* file inside of our testing location as shown: ***Reference Image 2-S.***

### Image 2-S

Graphical user interface, application

Description automatically generated

Once it looks like the image of ***Image 2-S,*** click next to continue. The next step will be to hit finish and viola; you have completed making your first automation which should run the Pull.py automatically: ***Reference Image 2-T.***

### Image 2-T

Graphical user interface, text, application, email

Description automatically generated

# Problems Encountered

## Roadblock One

* We are unable to execute bash commands from the Task Scheduler on any executable program inside of a S: drive folder via a batch file. Works for the local batch file on the C: drive and not the shared batch file on the S: drive. For example, ***Reference Image E-1;*** I have a locally stored batch file by the name of move.bat and I have the two file locations opened the one directly below my code is the local C: location which holds a single **DESPurple07122022.csv** file with a time stamp of 7/7/2022 3:45 pm. Yes, this is the wrong date and it is supposed to be because I just renamed it to prove my roadblock point. Back to what I was saying, the second file explorer below the top one is the S: drive location which shows multiple .csv files and one that is also **DESPurple07122022.csv.** However, this timestamp on this one is 7/8/2022 8:29 am. Lastly, the Task Scheduler to the right shows the file extension of the same batch file shown in this snippet. When ran with Task Scheduler; ***Reference Image E-2,*** you will now see that the file is missing locally on the C: drive and replaced the previous **DESPurple07122022.csv** on the S: drive. Meaning, I can write and move to the S: drive from my locally stored files on the C: drive. But when I repeat this process; ***Reference Image E-3,*** the file stays put and is never referenced, and we get an error code of 0x1 inside of the Task Scheduler. Which from what I have seen is a permission issue based on my role and the rights I have for executing certain files inside of the DES infrastructure. My code is not the problem because it worked with the local drive of C: and all I did was change arbitrary address destination names, which, are not incorrectly spelled because I copied them right from the command line.

### Graphical user interface, application Description automatically generatedImage E-1

### Graphical user interface, application Description automatically generatedImage E-2

Graphical user interface, text, application

Description automatically generated

### Graphical user interface, application Description automatically generatedImage E-3

### Why Do We Need This Roadblock Cleared?

* Our focus is a hybrid environment for this project, which means that we need a shared file location to pick up where the human/automation computer leaves off. Meaning, if we were to do all of this locally, we would be creating unnecessary .csv files and would be creating more work for ourselves because then I would need to create another automation program that would have to scalp the two or more .csv files together. Having this roadblock cleared would mean that now we could write and move inside of the S: drive AND the essential Pull.py could then work automatically with task scheduler. It is not shown here; but I tried it and it hit the same role/executable rights problem as the move batch file hit.