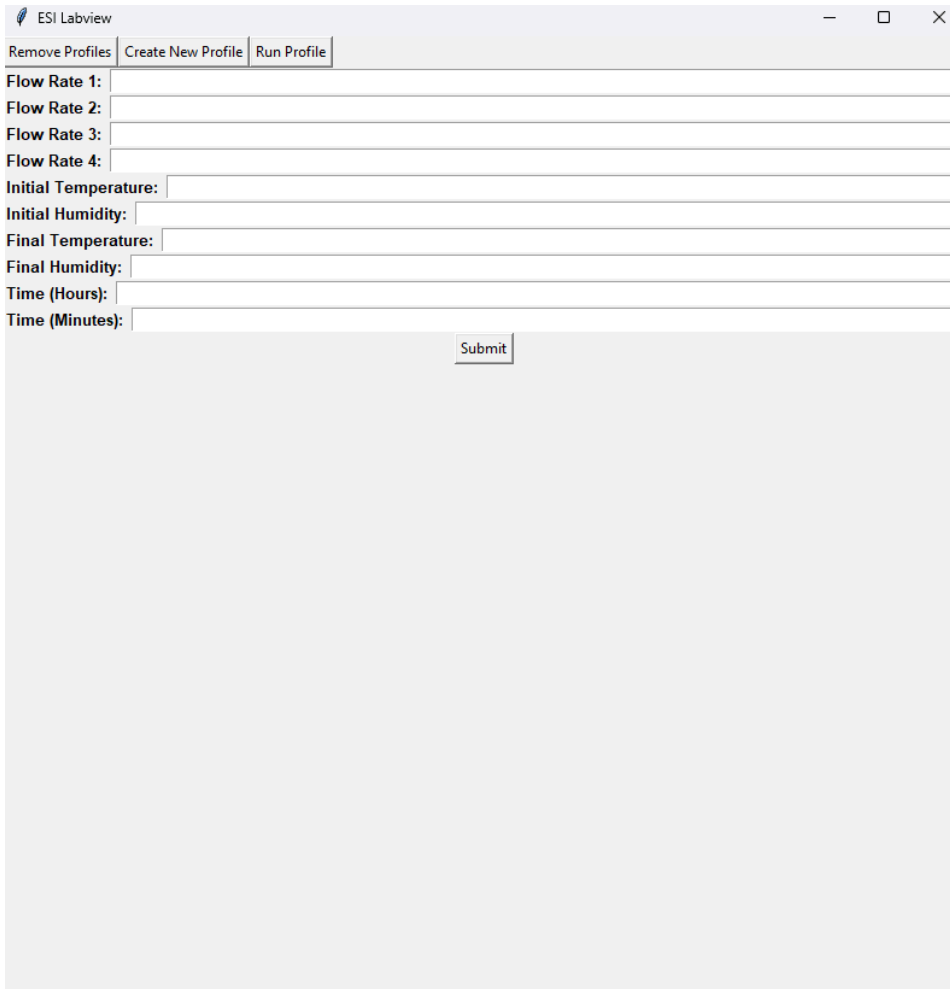


General Usage:

Using the tool is mostly quite intuitive and straight-forward. First, we'll go through each of the three pages within the application, and how to use them. To start the GUI, run main.py from the root directory.

Create a Profile:



ESI Labview

Remove Profiles Create New Profile Run Profile

Flow Rate 1:

Flow Rate 2:

Flow Rate 3:

Flow Rate 4:

Initial Temperature:

Initial Humidity:

Final Temperature:

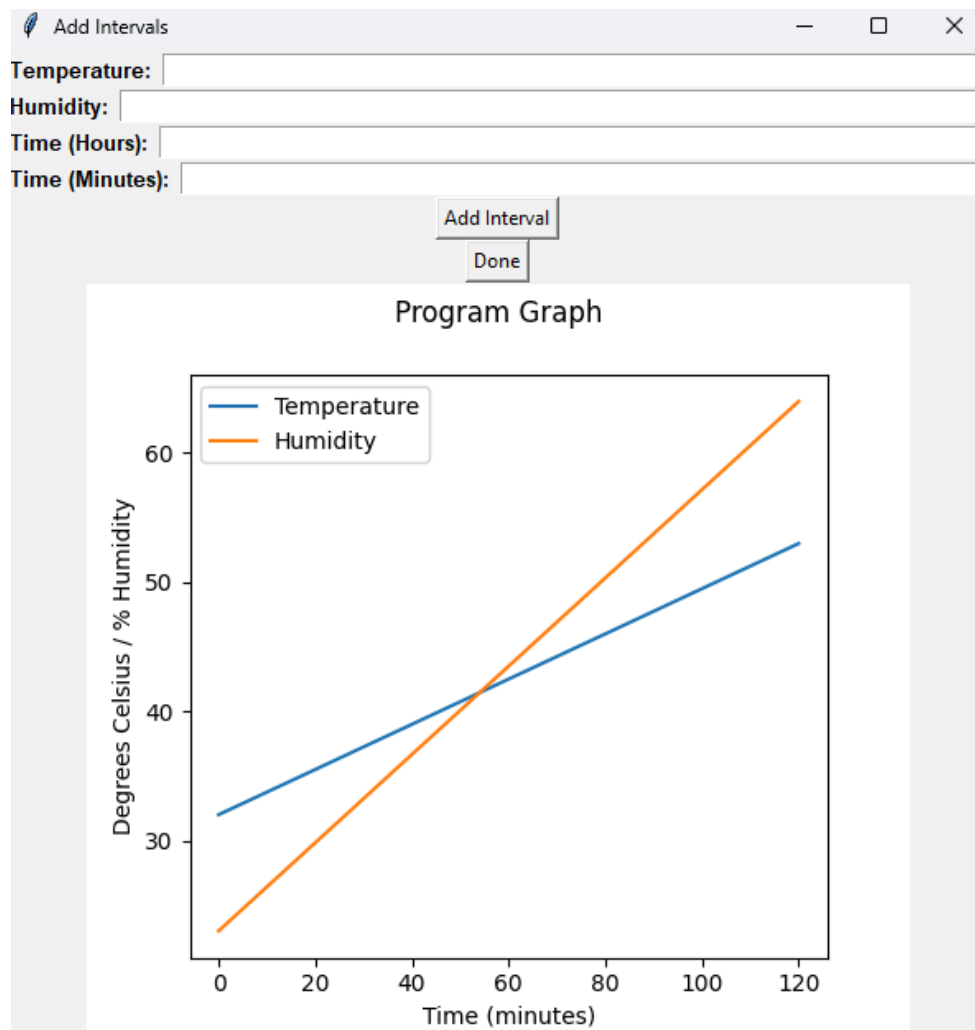
Final Humidity:

Time (Hours):

Time (Minutes):

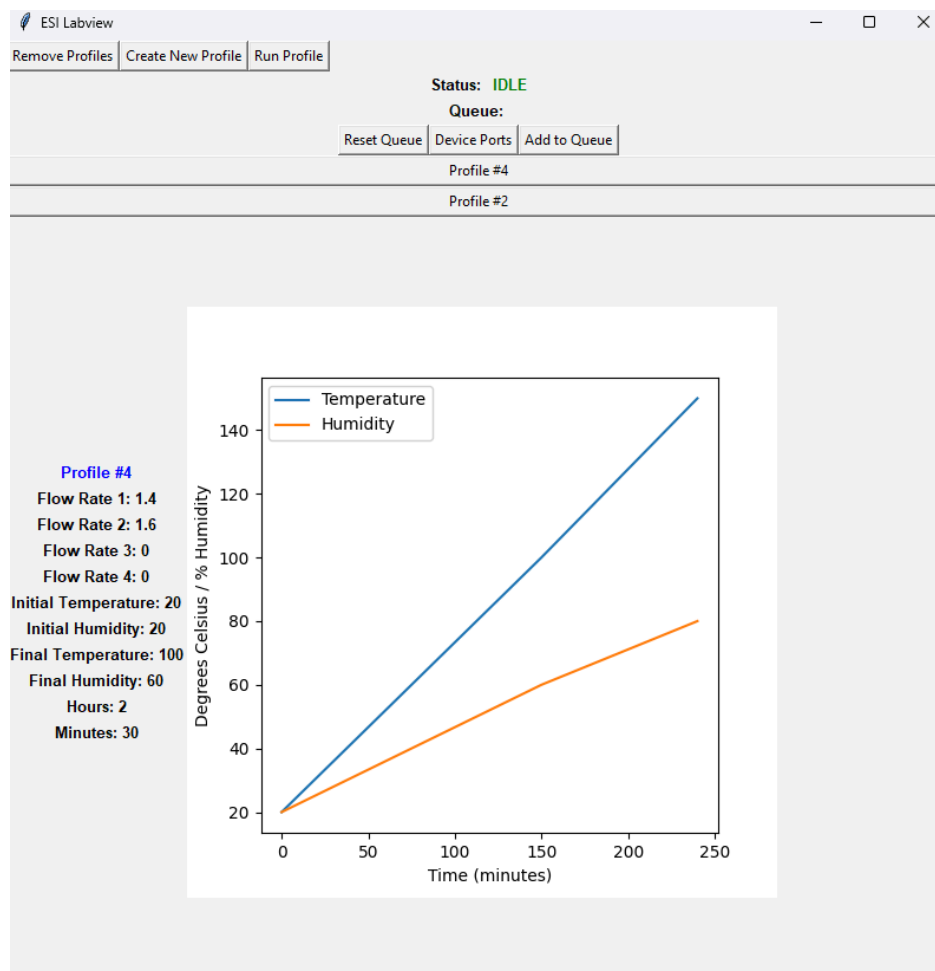
Submit

From this page, one can easily create new programs to run in the system. Since the Brook's MFC unit has the capability to control 4 MFCs, we've included 4 flow rate entries. However, only one of the Flow rates needs to be provided. All other variables must be provided (one of Hours or Minutes can be zero).



After pressing submit, a new window will appear, prompting you to add intervals of temperature and humidity for the Thermotron unit. Added intervals will make changes to the graph to provide a visual guide. Intervals are not required however, and this window can just be closed if so.

The profiles are saved in a CSV file in the same directory as the python source files. The user will never be required to manually edit this CSV, however, it can be used as a reference to see what has previously been run. The CSV can also be manually edited if needed.



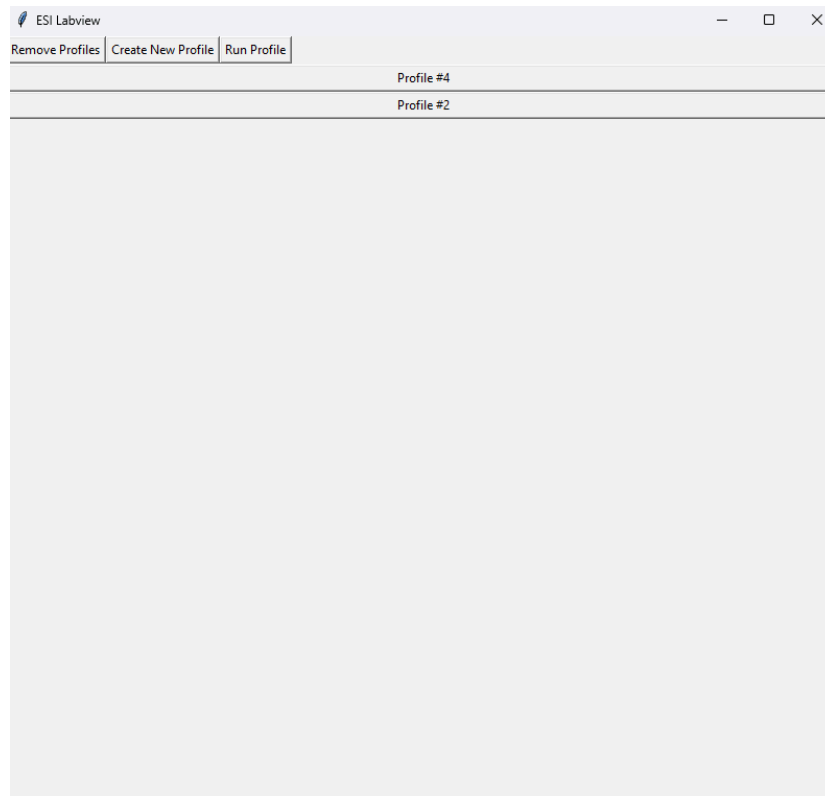
From the Run Profiles page you can execute programs to the entire system. This is done easily by selecting a profile from the list and adding it to the queue. Details of the selected profile are shown as well. Profile data is always parsed directly from the CSV, so the data will reflect any recently made changes to the profiles.

The screenshot shows a window titled "Device Ports" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are two input fields at the top: "Brooks MFC COM Port #:" and "Thermotron COM Port #:". Below these fields is a large, light gray rectangular area. In the center of this area, the text "Previously Saved Device COM Ports:" is displayed in green. Underneath this text, three lines of black text list saved ports: "Brooks MFC COM Port #: COM2", "Thermotron COM Port #: COM4", and "Sensor1 COM Port #: COM3". At the bottom of the window, there are three buttons stacked vertically: "Add Sensor", "Use Saved Ports", and "Use Newly Entered Ports".

Before submitting the queue and pressing run, the device ports must be set up properly to ensure the device communication is running smoothly. Upon running the application for the first time, the device ports will be saved locally so that in future runs you won't have to repeatedly enter in the port information. However, it is strongly recommended that you double check your ports before running each program as they can sometimes change.

Once your queue is established, simply press Run, and the application will transition into monitor mode. From there, you can also pause or stop the experiment at any time using the buttons on screen.

The sensor data is written to an output CSV file within the same directory. Additionally, if an email was provided when running the program, the sensor data will be emailed to you as well.



On the Remove Profiles tab, you can easily delete any of your saved profiles. It is important to note that **you can only have 10 saved profiles at once**. Once this limit has been reached you will not be able to create more profiles, and you will need to delete one.

Code Structure:

Each of the 3 pages of the GUI applications are coded within their own python source file. main.py initializes a tkinter container class that holds the 3 pages. A Page class is also created for each of the 3 windows to use as a base. Home.py contains the Remove Profile code, createProfile.py contains the Create Profiles code, and runProfile.py contains the Run Profile code.

Each of the devices within the system, the MFCs, Thermotron, and Sensors, each have their own python source files responsible for communication. A class is created for each device, and their objects are instantiated within runProfile.py to be used. Additionally, port_scanner.py is used to enumerate each port within the USB hub

Running an experiment:

Basic functionality:

Each experiment is defined by a profile. The program will cycle all devices accordingly through the defined profile(s). During the profile it will poll the Thermotron for temperature and humidity data and at the same time poll all the available sensors for their data.

Initial interval (Interval 0):

After defining a profile and all the necessary device setup the program will begin the initial state of running a profile (interval 0).

This initial interval will bring the Thermotron to the previously defined initial Temperature and humidity, this achieved by running the Thermotron in the manual mode. The thermotron will run in the manual mode until it has seen that it has reached the setpoints (within a certain allowable deviation) for a certain amount of time.

The below variables change the following respectively. Setpoint_ok_max is the amount of loops through the program it will wait before moving on to the next interval (larger number = more time). temp_deviation and humidity_deviation form a range of “setpoint +/- deviation value” of acceptable values.

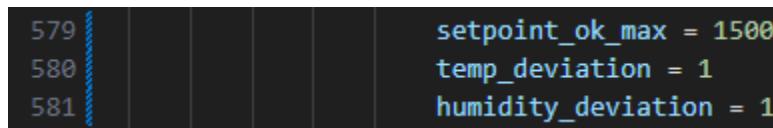
Example values:

```
setpoint_ok_max = 1000
```

```
temp_deviation = 2
```

```
humidity_deviation = 4
```

Result: The program will run in manual to the defined setpoints, until it has seen the temperature be between the setpoint +/- 2 and the humidity be between the setpoint +/- 4, for 1000 consecutive loops of the program



```
579 setpoint_ok_max = 1500
580 temp_deviation = 1
581 humidity_deviation = 1
```

(in the runProfile.py file)

All other intervals:

The rest of the intervals will be executed in the program mode of the thermotron. Based on the user defined profile from the GUI it sends all the necessary commands to write that program to the thermotron (please read the section in the thermotron manual on writing and running programs to understand this).

Once the program is written to the thermotron, the python script waits and monitors the operation of the thermotron until an error occurs or the profile is completed. The program will poll the data from the sensors and the thermotron and write them to a csv. This csv will be named:

“Thermotron_data_YYYY-MM-DD_HH_MM_Program_Number_#.csv”

Where YYYY-MM-DD is the date the program started, HH_MM is the time in hours and minutes the program started, and # is whichever profile was run. (This file will be emailed to you if an email was entered in the GUI, if the program encounters any errors it will email you this as well, we strongly recommend entering an email every time). The CSV will be saved in a folder named “Thermotron_data” located in the same location as the python script.

Email information:

The Thermotron will send emails from “thermotronabb108@gmail.com”. This is defined in the Thermotron.py file (at the bottom) by the following 2 variables.

```
361 email_sender = 'thermotronabb108@gmail.com'
362 email_password = 'cjum zibl qwyd oecn'
```

The password is not the same password you use to login, but this password is what allows python to use this email to send messages.

If you want to use a different email as the sender, change the email_sender variable to the account you're using. You will need to allow python to access this email, by generating a unique password for it. Below is a link to a quick guide on how to generate the password.

Credentials to log in to the gmail account:

Email: thermotronabb108@gmail.com

Password: @Thermoabb108

https://www.youtube.com/watch?v=zxFXnLEmnb4&ab_channel=CodeWithTomi

Note: Please use a gmail account, or else more lines in the code would have to be changed, specifically below.

```
386 with smtplib.SMTP("smtp.gmail.com", 587) as server:
```

The “smtp.gmail.com” and “587” would have to change based on whichever service you use for your email.