CLOUD SENSOR MANUAL

This is an overview into starting and troubleshooting the cloud sensor we built for our senior project. We will cover how to start taking data as well as editing sections of the sensor.

Jake Tillman & Corey Curran
jakeman412@gmail.com, ccurran97@gmail.com

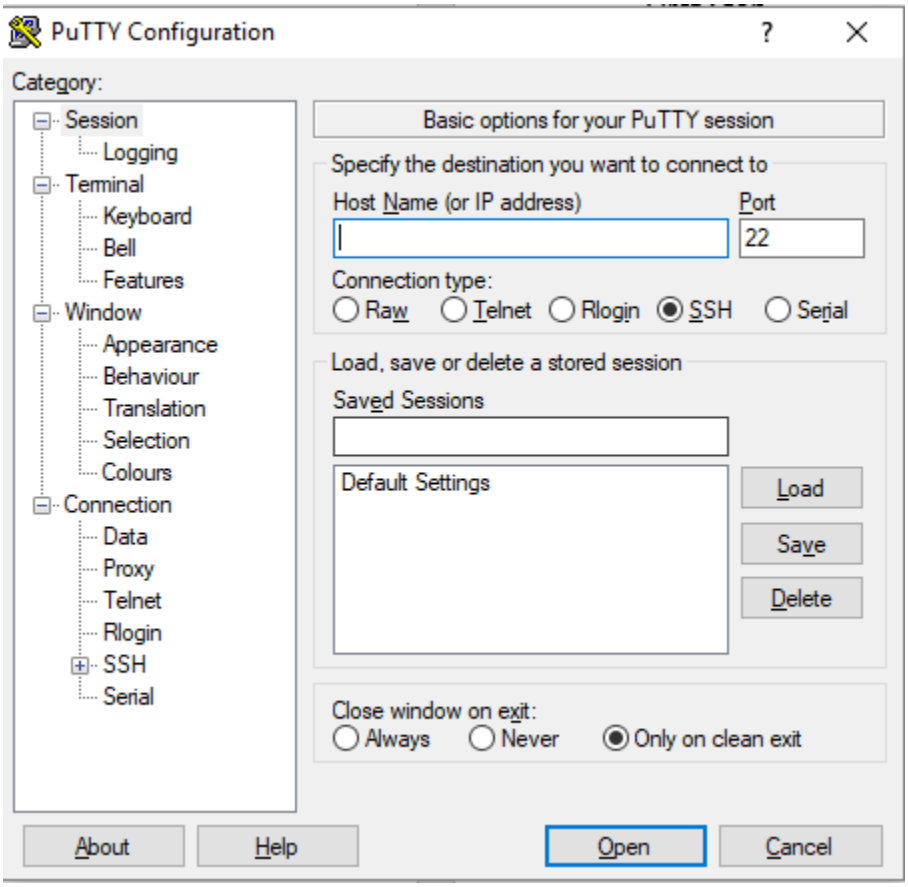
Contents

Start-up Guide

First Step

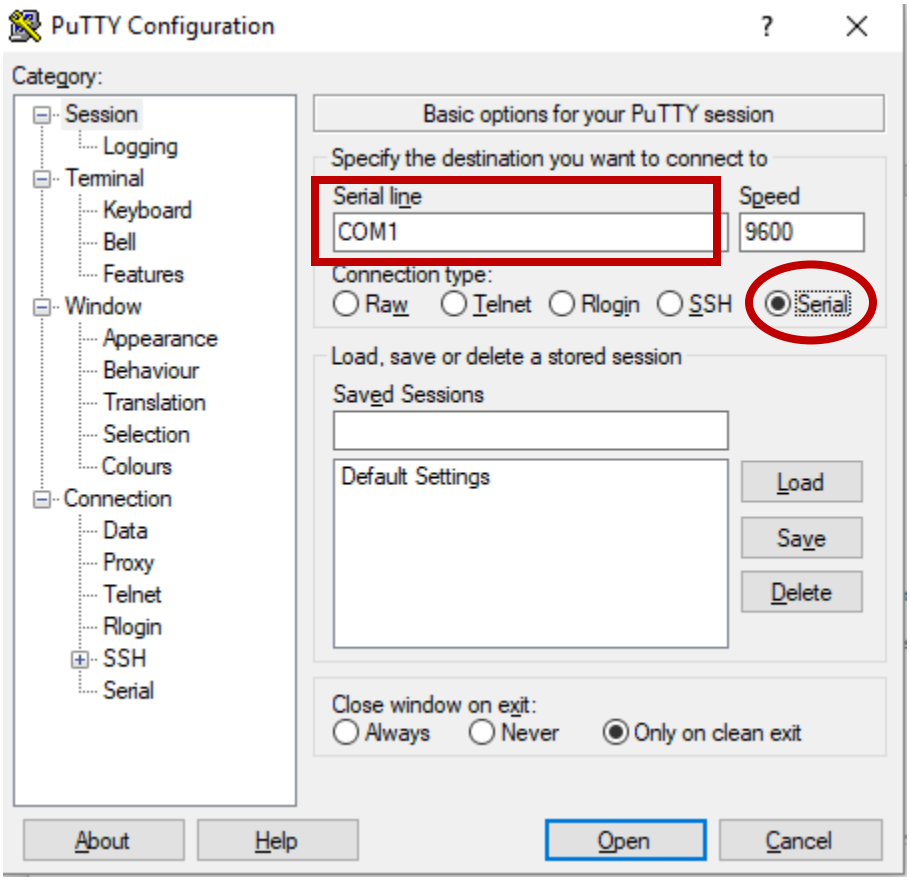
Open the putty application on your computer

The start screen looks like this

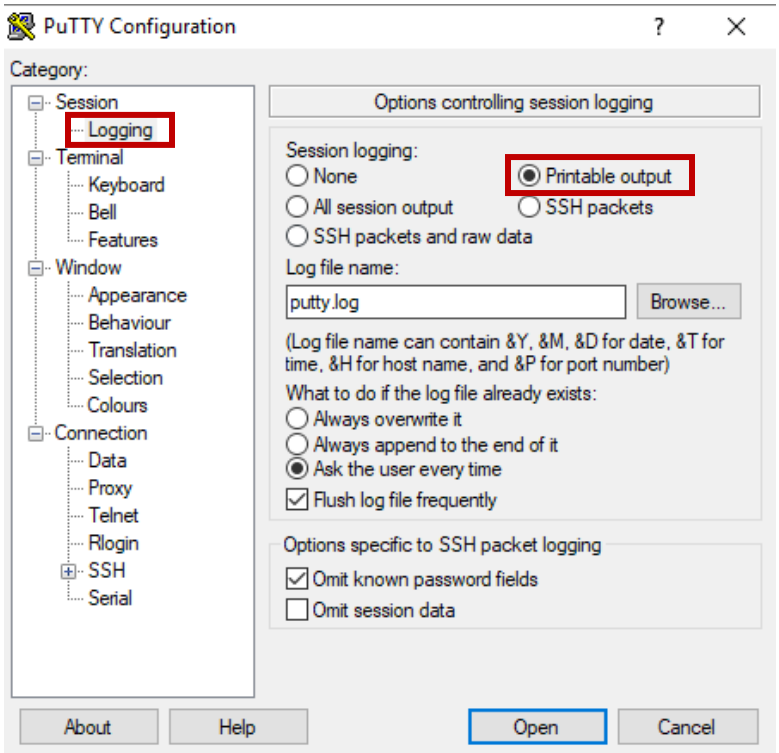


Make sure Serial is selected and type in what COM port the device is using

(if you don't know try COM3, COM5, or COM7, etc..)

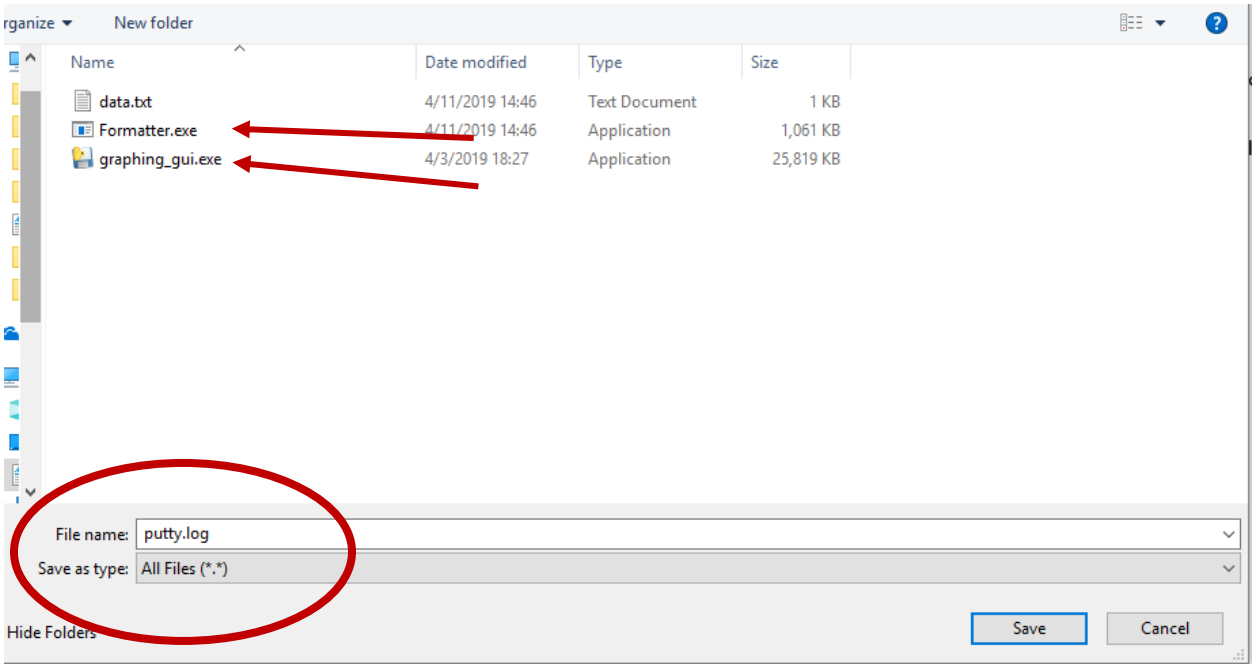


Next step is to select Logging and Printable output



Now we need to save the log file. Go ahead a select Browse and choose where you want the log file to be stored at. **DO NOT CHANGE THE NAME OF THE FILE**

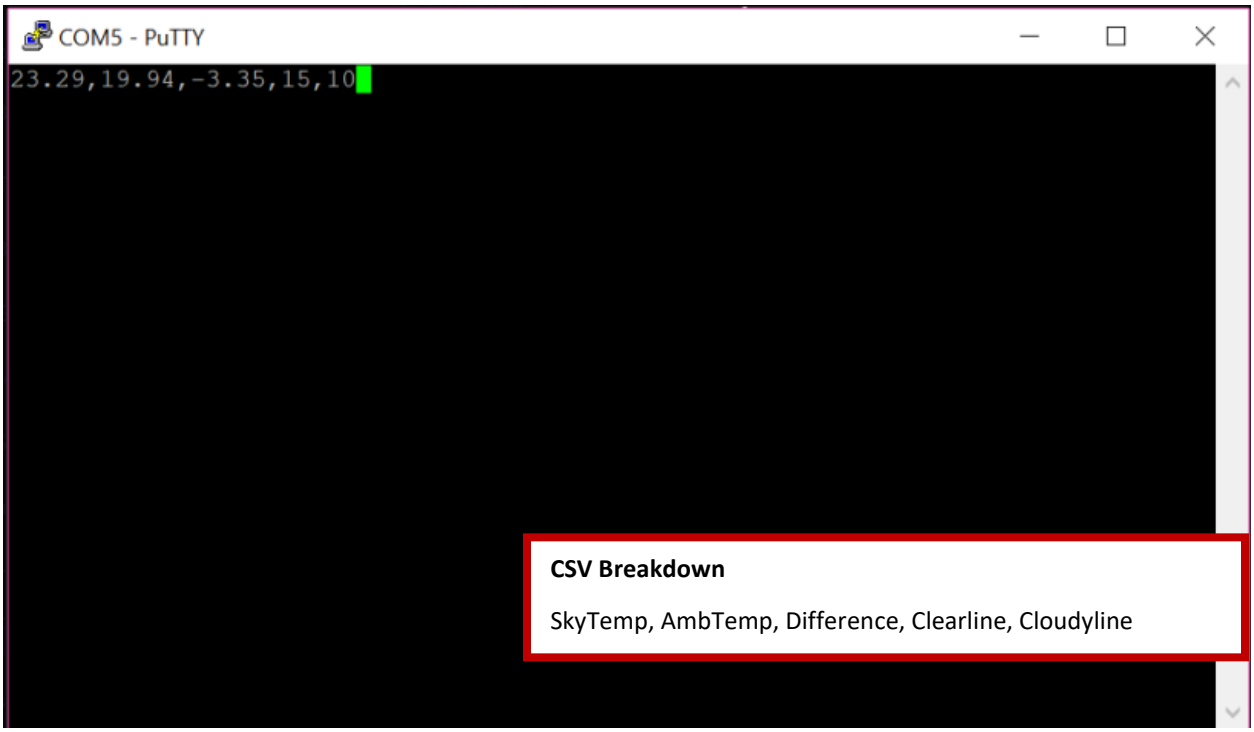
Note: This log file must be stored where the graphing GUI is as well as the formatter script.



Now all you should have to do is hit save and then go back to putty select Open

Note: If you have done this before and there is already a putty.log file at that location then a new window will open asking you if you want the file wiped or overwritten. Simply select the option you want.

You should see a screen similar to this (the numbers will be different most likely).



Now as long as you have that black putty window open and running our device will continue to take data and log it into that putty.log file. Our Graphing GUI and C script will take that data and plot or format it accordingly.

Once you close the putty window you may delete or archive the log file.

NOTE: Once the putty window is closed our device will stop taking data and the Graphing GUI as well as the Formatter (C script) will stop updating and just keep reading the last value that the putty.log file holds. So, if you want the cloud sensor to work you must have putty open.

Starting Graphing GUI

As long as the **putty.log** file is **located** where **the graphing_gui.exe** is located at simply click on the graphing_gui.exe to start it.

Here is a link to the GitHub where I have placed the code for the C script and Graphing GUI

(https://github.com/jtillman4/cloud-sensor-project)

Starting Formatter

The formatter is used to change the CSV file into the Boltwood cloud sensor 1 output file. This is used so that WeatherWatcher can understand what our device is relaying to the computer. All for the observatory to become automated eventually. The formatter is written in C++ and simply reads in the putty.log file and creates a new file called data.txt that WeatherWatcher will read and understand.

This format is provided for compatibility with software written to use the original Cloud Sensor product.
The format is:

```
Date       Time     U   SkyT  AmbT  SenT Hea W Since  Now() Day's S
2005-06-03 02:07:23 C  -28.5  18.7  22.5   3 0 00004 038506.08846 1
```

The immediately above line is an example of the only line in the file.

The header line is here just for illustration. It does not actually appear anywhere.

The fields mean:

Date	local date yyyy-mm-dd
Time	local time hh:mm:ss (24 hour clock)
U	temperature units, 'C' for Celsius or 'F' for Fahrenheit
SkyT	sky-ambient temperature
AmbT	ambient temperature
SenT	sensor case temperature
Hea	heater setting in %
W	wet flag, =0 for dry, =1 for wet
Since	seconds since last valid data
Now() Day's	date/time given as the VB6 Now() function result (in days) when Clarity II last wrote this file
S	sky condition (see the SkyCond enum below).

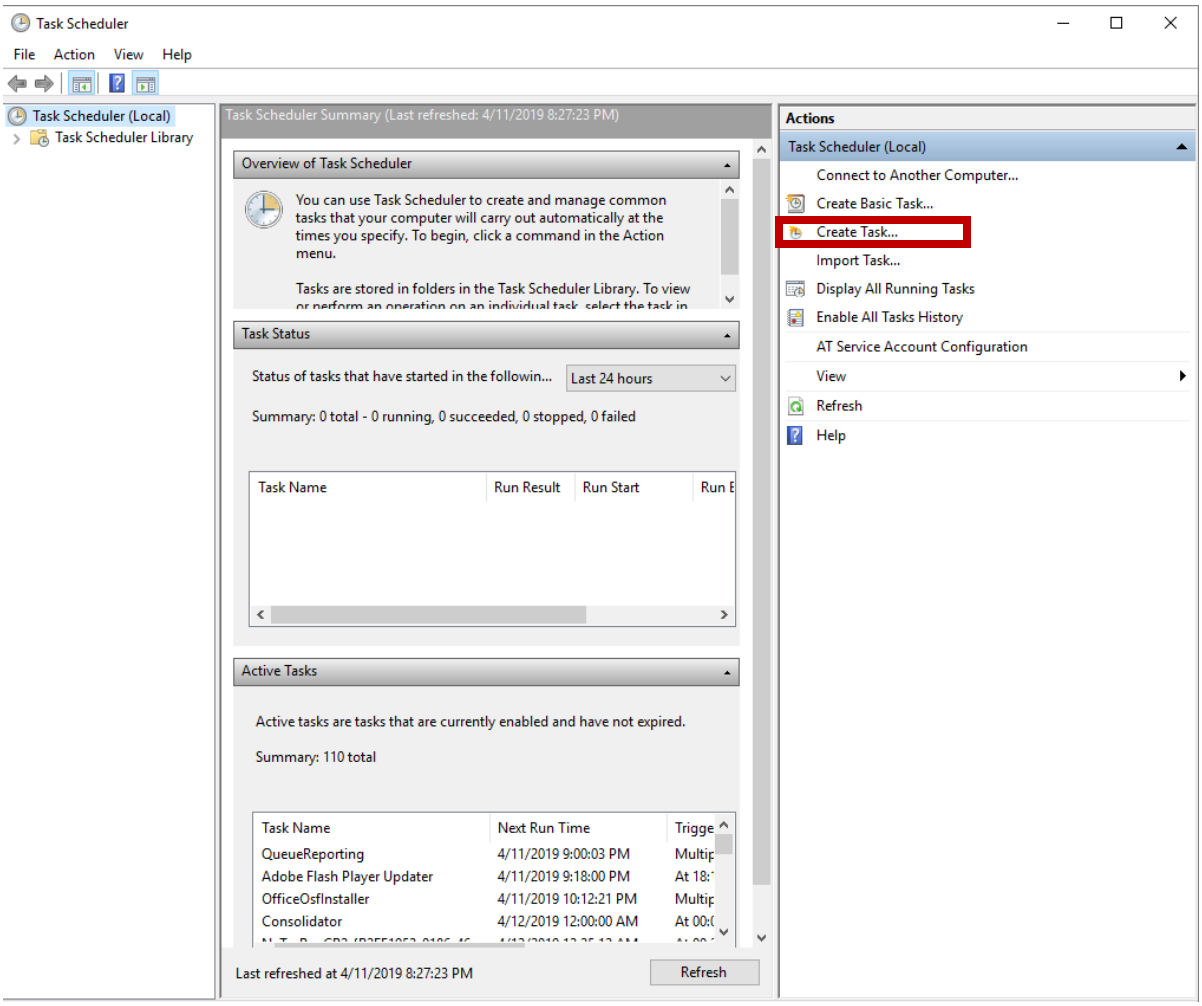
```
Public Enum SkyCond
   skyUnknown = 0
   skyClear = 1
   skyCloudy = 2
   skyVeryCloudy = 3
   skyWet = 4
End Enum
```

This is from the Boltwood Cloud Sensor Manual and this is the format the formatter is copying to run WeatherWatcher.

To make the Formatter run we are going to use windows Task Scheduler

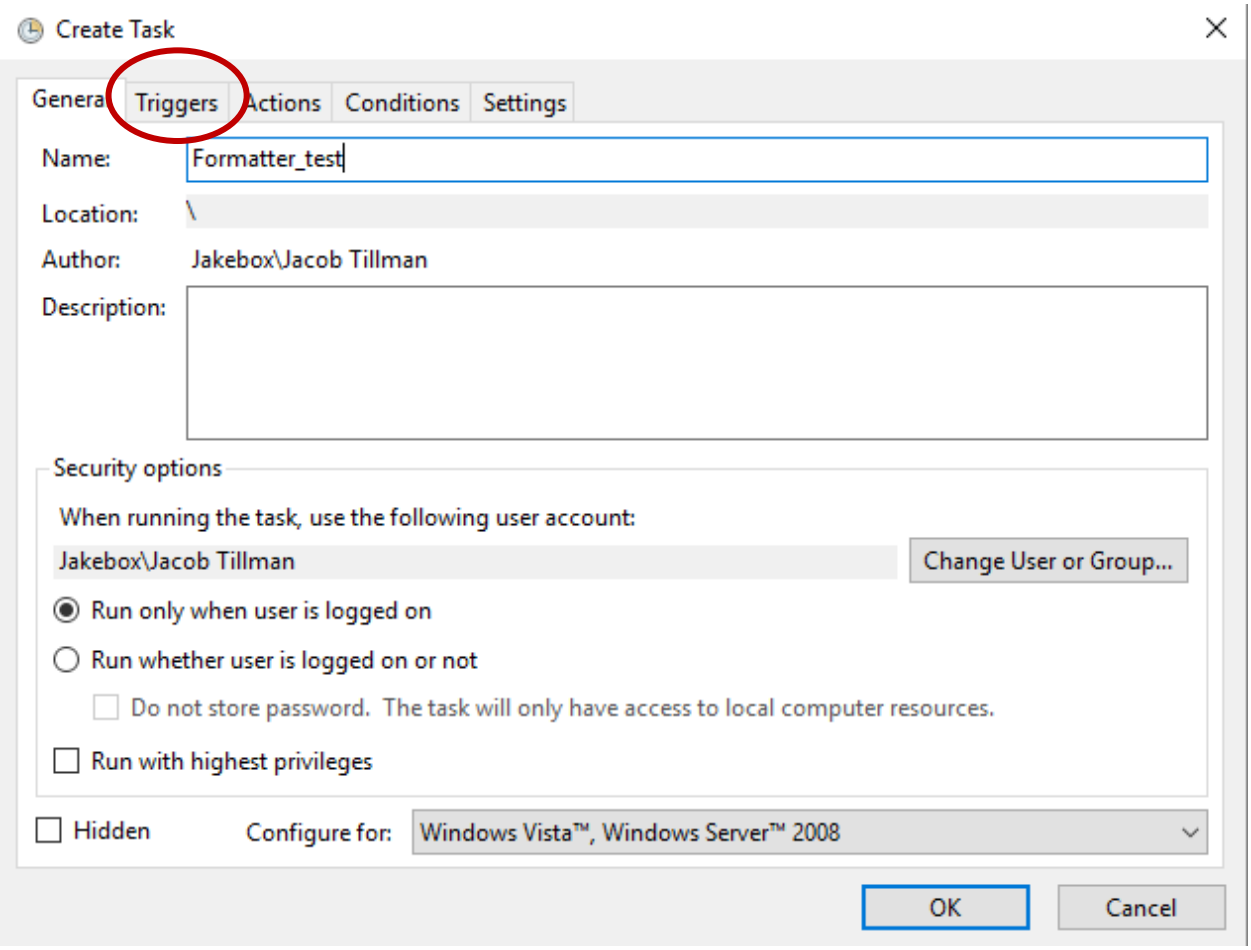
NOTE: If there is not a putty.log file in the same location as formatter it will not work.

First open Windows Task Scheduler. You should see a screen like this.



Go ahead and click Create Task

Once in the task editor you must name your task. You may call it whatever you like. For ease let's call it Formatter_test.

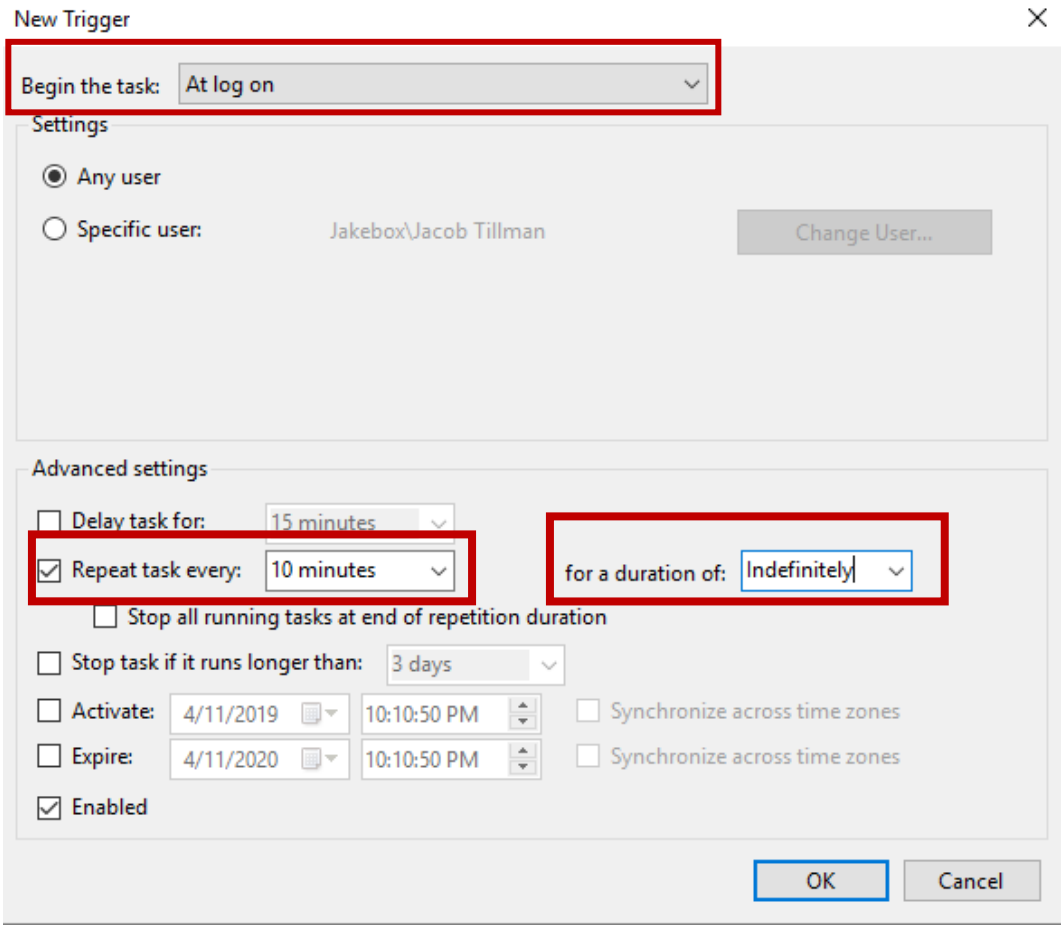


Now select the Triggers tab (highlighted above)

Go ahead and click New

In the triggers section we will pick how often Windows will run our formatter.

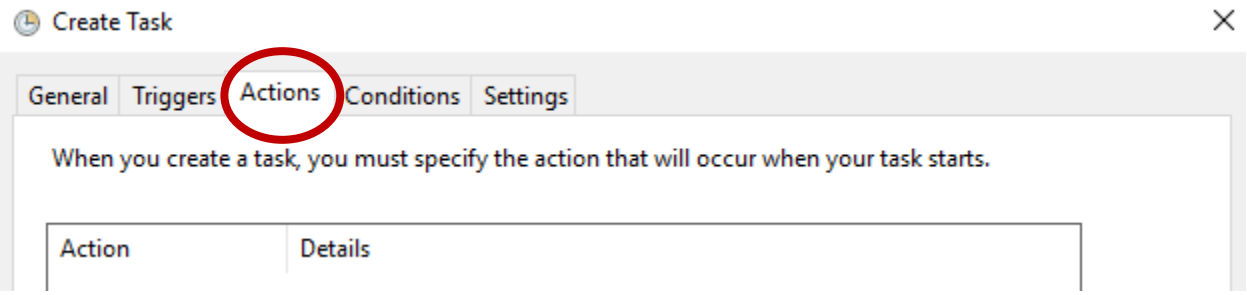
On the New Trigger window first choose begin the task: At log on



Then make sure you have repeat task every 10 minutes (or however often you want the formatter to run for WeatherWatcher). Also make sure you have a duration of Indefinitely set.

Go ahead and click Ok now.

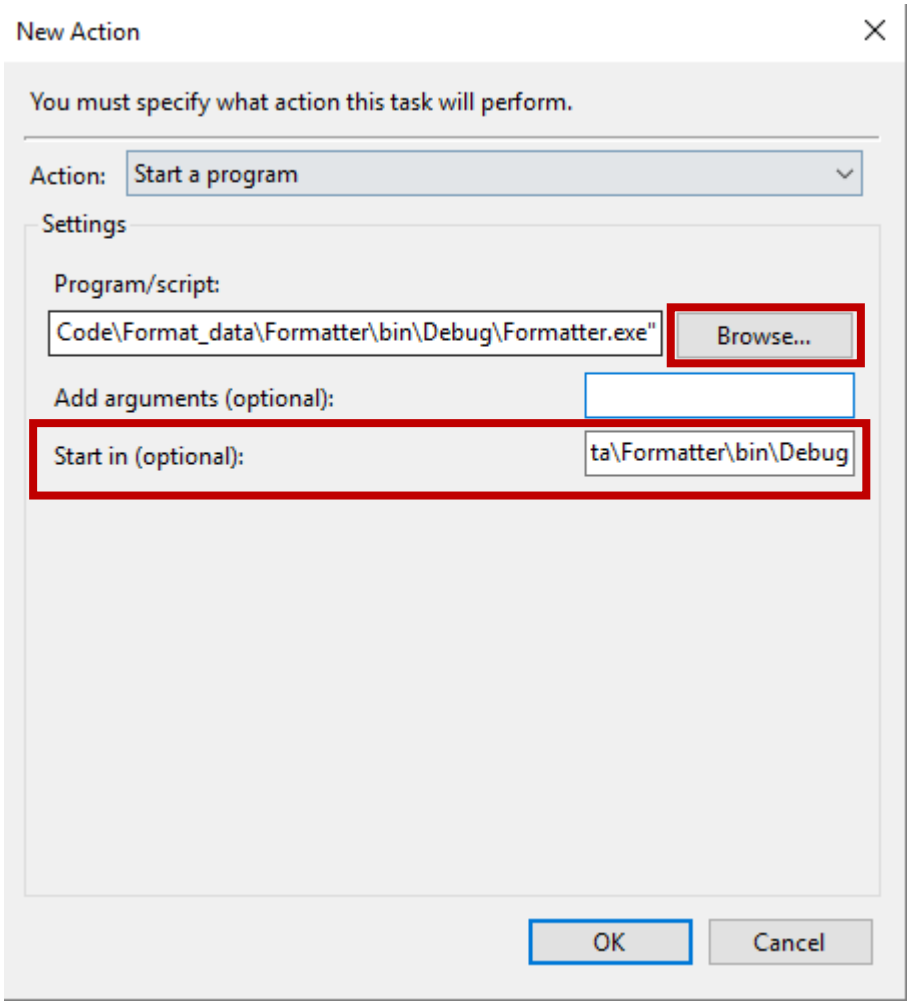
Now we are back to the task editor and click on the Actions tab.



Now click on the New button on the bottom.

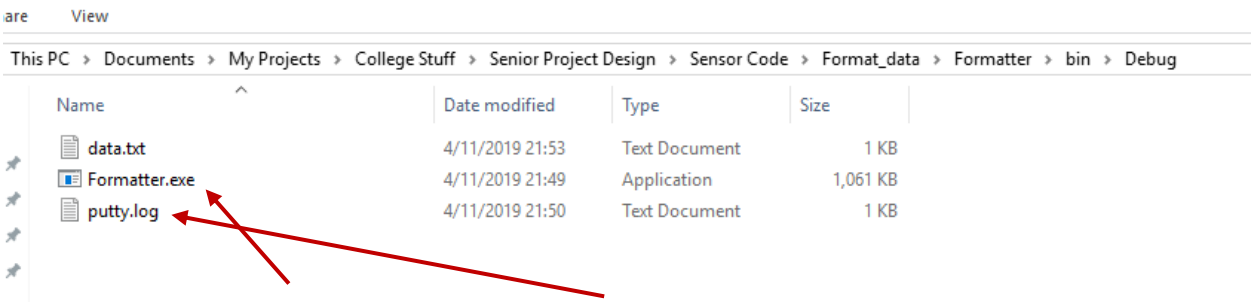
In the Action window you must choose the .exe file you wish to run.

In this case we want to run Formatter.exe. Select Browse and locate the Formatter.exe.



Make sure to also paste the pathname of the folder Formatter.exe (& putty.log) are located in as the Start in option

In my case the folder I am using looks like this.



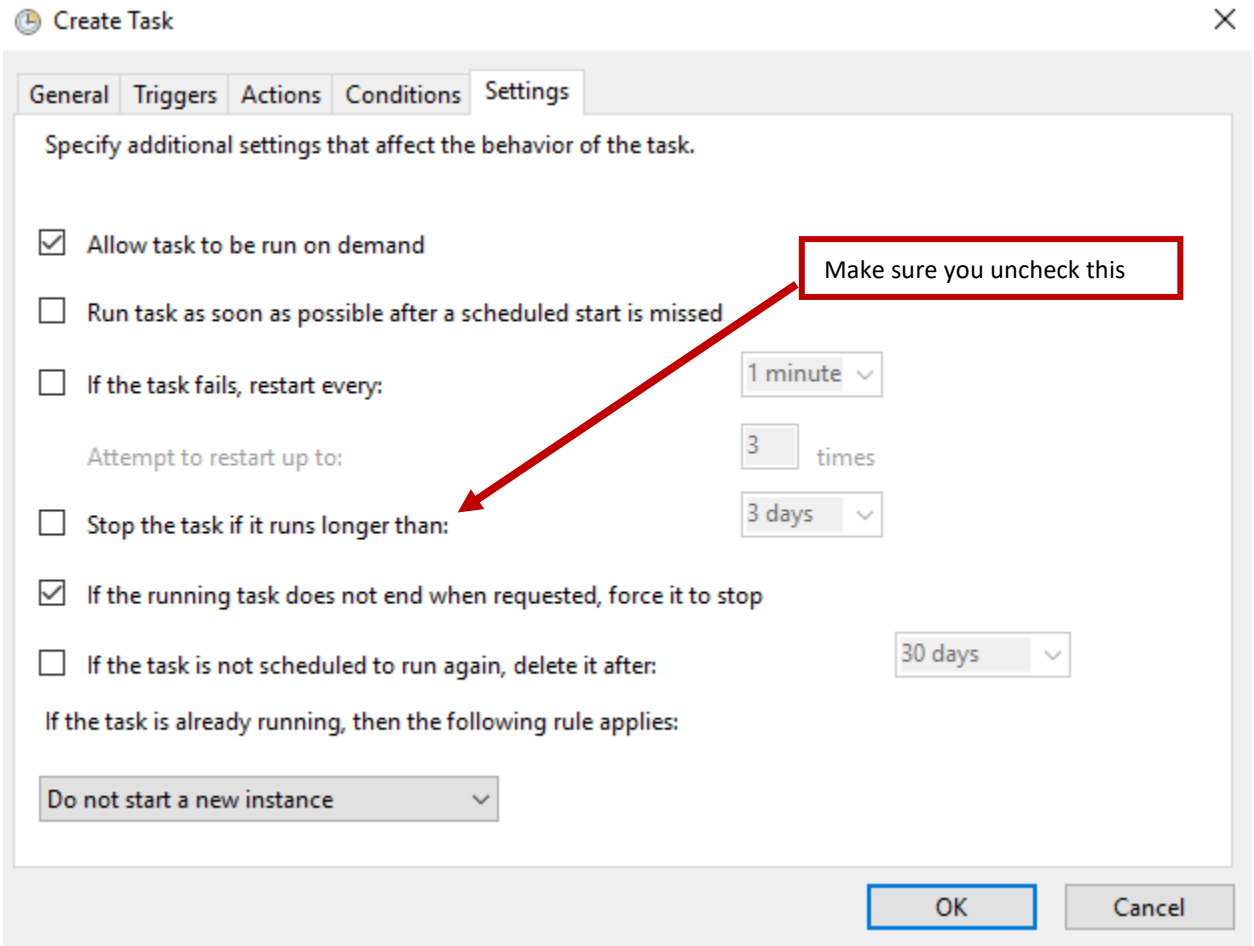
It has both the Formatter.exe as well as the putty.log file.

So, I use the pathname C:\Users\Jake Tillman\Documents\My Projects\College Stuff\Senior Project Design\Sensor Code\Format_data\Formatter\bin\Debug

NOTE: There should be no quotes around the pathname in the Start in box. There will however be quotes around the one in the Program/Script Box.

After filling this out select Ok.

Now go to the Settings tab and check the boxes I have.



Now go ahead and click Ok and you should be back at the Task Editor. Select Ok and now your Windows task is scheduled. Since we chose at log on the task won't be running at this point in time. So, if you want the Formatter to start running right away you should restart/log back in.

If all has been done correctly every 10 minutes the data.txt file (that is located at the same location of Formatter.exe and putty.log) should update and change depending on the device data changed.

NOTE: If you have done this before you shouldn't have to do this every again. Windows should just automatically schedule this to be done every 10 min like you specified. Meaning data.txt gets updated every 10 min which means WeatherWatcher will update every 10 min.

Arduino Code Overview

```
#include <Wire.h>// I2C library, required for MLX90614
#include <OneWire.h> //Library for waterproof sensor
#include <DallasTemperature.h> //Library for waterproof sensor
#include <SparkFunMLX90614.h> // SparkFunMLX90614 Arduino li

#define ONE_WIRE_BUS 2 // Data wire is plugged into pin 2 on the Arduino

// Setup a oneWire instance to communicate with any OneWire devices
// (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);


IRTherm therm; // Create an IRTherm object to interact with throughout



void setup() {
 Serial.begin(9600);

 therm.begin(); // Initialize thermal IR sensor
 therm.setUnit(TEMP_C); // Set the library's units to Farenheit
  // Alternatively, TEMP_F can be replaced with TEMP_C for Celsius or
  // TEMP_K for Kelvin.


 sensors.begin();
}

void loop() {
   //Enter limits for clear and cloudy here in Celcius.
    /*
     *
     *
     *
     */
    float ClearLimit = 15;
    float CloudyLimit = 10;
    /*
     *
     *
     *
     */
     //Do not change code under this line!


 if (therm.read()) // On success, read() will return 1, on fail 0.
  {
    Serial.print(String(therm.object(), 2)+",");

  }
/*****************************************************************/
 //Serial.print("Waterproof Sensor: ");
 sensors.requestTemperatures(); // Send the command to get temperature readings
/*****************************************************************/
 //Serial.print("Waterproof Sensor: ");
 Serial.print(String(sensors.getTempCByIndex(0)) + ",");

 Serial.print(sensors.getTempCByIndex(0)-therm.object()); //This the the difference of the 2 sensors
  Serial.print(",");
  Serial.print(ClearLimit);
  Serial.print(",");
  Serial.print(CloudyLimit);
   delay(600000);
 Serial.println();

}
```

This is a picture of the Arduino code that the Arduino UNO reads and executes.

For the clear and cloudy limits feel free to edit these two values. These will be read and cause updates to the graphing GUI and C script that WeatherWatcher reads.

This is how fast the device takes sample data in milliseconds.

Explaining the Code

This Arduino code is a basically summed up as follows.

Set the units for the IR sensor then start the sensor. We have provided two integer values that you can edit at your discretion to change the "clear" and "cloudy" limits. The code then continues into printing the value of the IR sensor (sky temp) on a serial monitor and places a comma after it. Then follows the waterproof temperature sensor (ambient temp) with a comma after it. Then follows a simple subtraction of ambient minus sky printed along with a comma. Finally, our clear and cloudy limits get printed. This is following simple CSV standard format. At the very end we have included a delay of 600000 ms which is 10 minutes. Feel free to change this. This will not change how often the other programs update though.

Libraries

The Arduino code we wrote works off 4 main libraries you will need to have installed into your Arduino integrated development environment. I have provided links to download the zip on each one.

1. Wire.h – This library is used for the IR sensor to read values (MLX90614) (should be included in base Arduino IDE)
2. OneWire.h – This library is used for the waterproof temperature sensor (DS18B20) (https://github.com/PaulStoffregen/OneWire)
3. DallasTemerature.h – This library is used for the temperature sensor call functions (DS18B20) (https://github.com/milesburton/Arduino-Temperature-Control-Library)
4. SparkFunMLX90614.h – This library is used for the IR sensor so Arduino can recognize it (MLX90614) (https://github.com/sparkfun/SparkFun_MLX90614_Arduino_Library)

To install these libraries simply click on the **sketch** drop down menu at the top of Arduino IDE. Next hover over **include library** this will show a new mean and near the top of that click on **Add .zip Library** then simply select the .zip files you downloaded from the GitHub links I have provided above.

NOTE: You only have to include the libraries once to your Arduino IDE once you have included them they should always be present, and you won't have to run this process again.

Editing the Code

Now one of the things Dr. Robertson asked us to provide him was a way to change the limits of what we consider “clear” or “cloudy” on the software side. Our solution was to add extra values to our CSV format in Arduino. The two values you can change and will be reflected in our graphing GUI and c script that WeatherWatcher reads. Once you have changed these values you should be able to simple hit the verify button in the top left of Arduino (looks like a checkmark) and once that’s done hit the compile button (looks like an arrow also in the top left).

Trouble Shooting Arduino

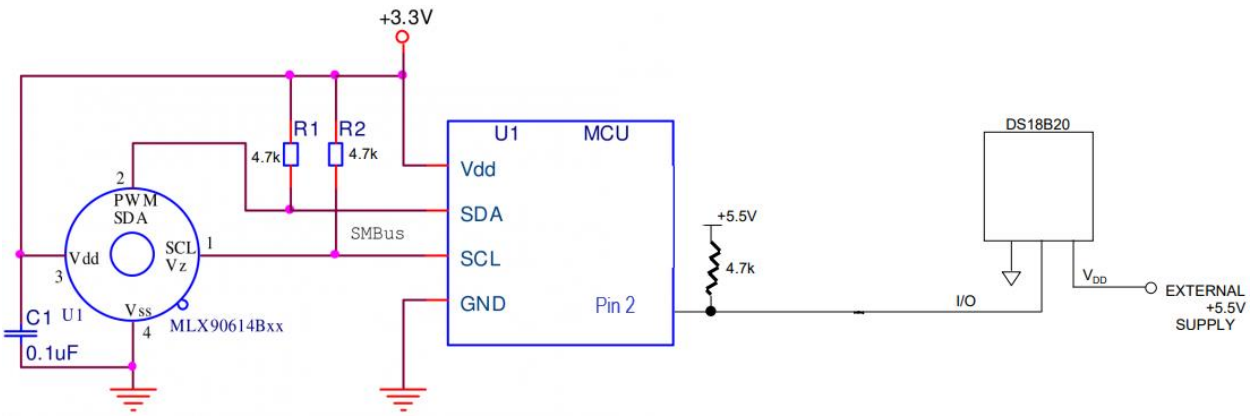
Make sure you have installed the libraries as I have directed and specifically the ones I linked.

Make sure to have the right COM port on Arduino under Tools>Port>COMx (ex. COM1, COM3, COM5, … etc) You will just have to test which one is the correct one because it can change depending on which USB port the device is plugged into.

The code for Arduino is also on my GitHub so you can reference it when you need to.

(https://github.com/jtillman4/cloud-sensor-project)

Hardware:



Parts:

Infrared Thermometer - MLX90614
Waterproof Temperature Sensor - DS18B20
3 pull up resistors (4.7kΩ)
0.1uF capacitor
Arduino Uno
30-foot usb cable
Black project box

How it is connected:

IR Sensor (MLX90614):

MLX90614 Pin	Arduino Pin	Note
VDD	3.3V	Nearby 0.1µF decoupling capacitor two 4.7kΩ pull up resistors
VSS	GND	Nearby 0.1µF decoupling capacitor
SDA/PWM	SDA	Pulled up to 3.3V via a 4.7kΩ resistor
SCL	SCL	Pulled up to 3.3V via a 4.7kΩ resistor

Vdd is 3.3V. This is connected to the VDD pin on the MLX90614.
The 3.3V Vdd line is also connected to the 0.1uF capacitor. The capacitor is then connected to ground.
The Vss pin on the MLX90614 is also connected to ground.
The SDA pin on the Arduino is connected to the SDA/PWM pin on the MLX90614. There is also a pull up resistor of 4.7kΩ connected to the SDA line to the 3.3Vdd line.
The SCL pin on the Arduino is connected to the SCL pin on the MLX90614. There is also a pull up resistor of 4.7kΩ connected to the SCL line to the 3.3Vdd line.
Use the notch on the MLX90614 to help identify which pin is which.

Water Proof Sensor (DS18B20):

DS18B20 Pin	Arduino Pin	Note
VDD (Red wire)	5V	4.7kΩ pull up resistors
GND (Black wire)	GND	Goes to ground
DATA (White wire)	Digital Pin 2	Pulled up to 5V via a 4.7kΩ resistor

Vdd is 5V. This is connected to the Vdd pin on the DS18B20.
The GND pin on the DS18B20 is connected to ground.
The Data pin on the DS18B20 is connected to Digital Pin 2 on the Arduino. There is a pull up resistor of 4.7kΩ connected from the data line to the 5Vdd line.

Maintenance:

To clean the sensor **do not press down on the IR sensor itself.** Instead gently wipe over the sensor using a paper towel or use compressed air to clean the IR sensor from dust.
If bird droppings land on the IR sensor use a sponge with warm soapy water and gently wipe away the droppings using a paper towel to dry the sensor.
If snow or ice accumulates gently push / scrap off from the top of the sensor.

Troubleshooting:

If putty log is outputting -120 sky temperature -120 ambient temperature and a 0 difference (ex: -120 -120 0 10 15). Try unplugging the USB port waiting 10 seconds and plugging back in repeat a couple of times if first unplug does not work.

If the output is cloudy or very cloudy when it is clear, check the sensor for debris such as leaves, dust, or bird droppings. (Refer to Maintenance on how to clean IR sensor.)

Sources/References:

https://www.sparkfun.com/products/11050#comment-52a60f8cce395f052a8b456a
https://learn.sparkfun.com/tutorials/mlx90614-ir-thermometer-hookup-guide#hardware-hookup
https://www.sparkfun.com/products/9570
https://create.arduino.cc/projecthub/TheGadgetBoy/ds18b20-digital-temperature-sensor-and-arduino-9cc806
https://www.mta.ca/uploadedFiles/Community/Academics/Faculty_of_Science/Physics/Gemini_Observatory/Observing_Resources/cloud-sensor-II-users-manual.pdf
https://github.com/jtillman4/cloud-sensor-project
DS18B20 Data sheet