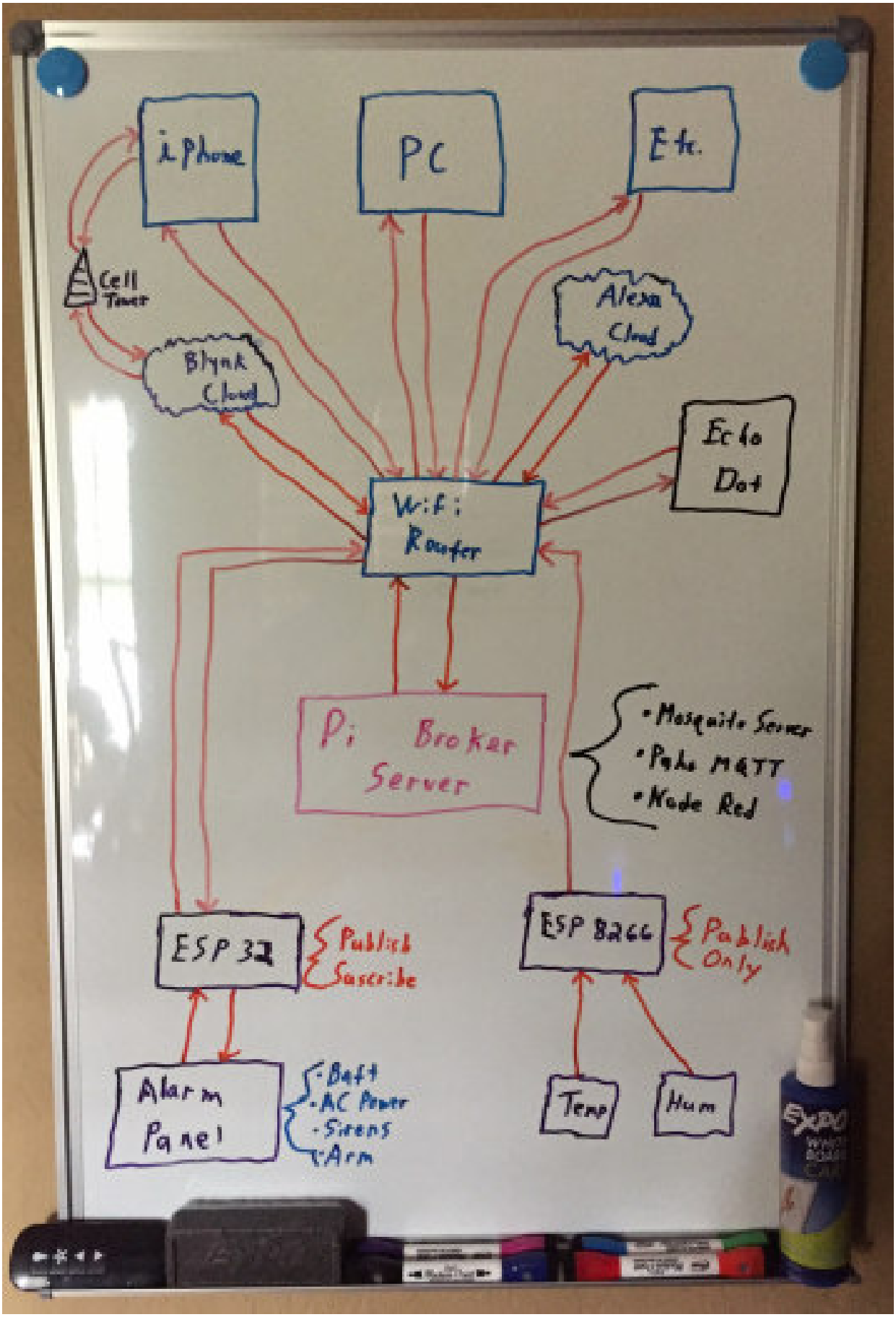
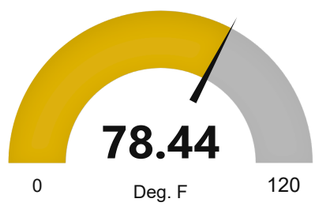

iPhone
PC
Etc.
Cell Tower
Alexa Cloud
Blynk Cloud
Echo Dot
Wi-Fi Router
Pi Broker Server
• Mosquito Server
• Paho MQTT
• Node Red
ESP 32
Publish Subscribe
ESP 8266
Publish Only
Alarm Panel
- Batt
- AC Power
- Sirens
- Arm
Temp
Hum


Node Red Controls
CIGAR HUMIDOR HUMIDITY
52.9
0
100
CIGAR HUMIDOR TEMP
78.4
0
120
HUMIDITY SETPOINT
45.2
ALARM BATTERY
12.0
0
14
ALARM AC POWER
120.0
0
140
ARMED
ALARM
SIREN
HORN
ALARM SIREN DIABLE
OFF
ALARM HORN DISABLE
OFF

Home

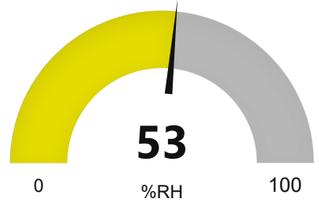
Temperature (Degrees F)

Cigar Humidor Temperature



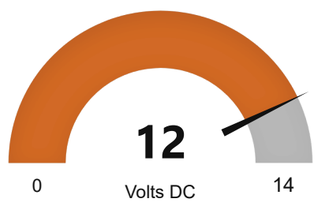
% Relative Humidity

Cigar Humidor Humidity



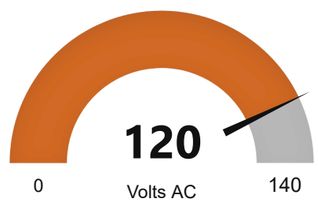
Battery Volts

Alarm Panel Battery Voltage



AC Power

Alarm Panel AC Volts



Alarm

Alarm Message **Off**

Armed

System Armed Message **On**

Alarm Horn Switch

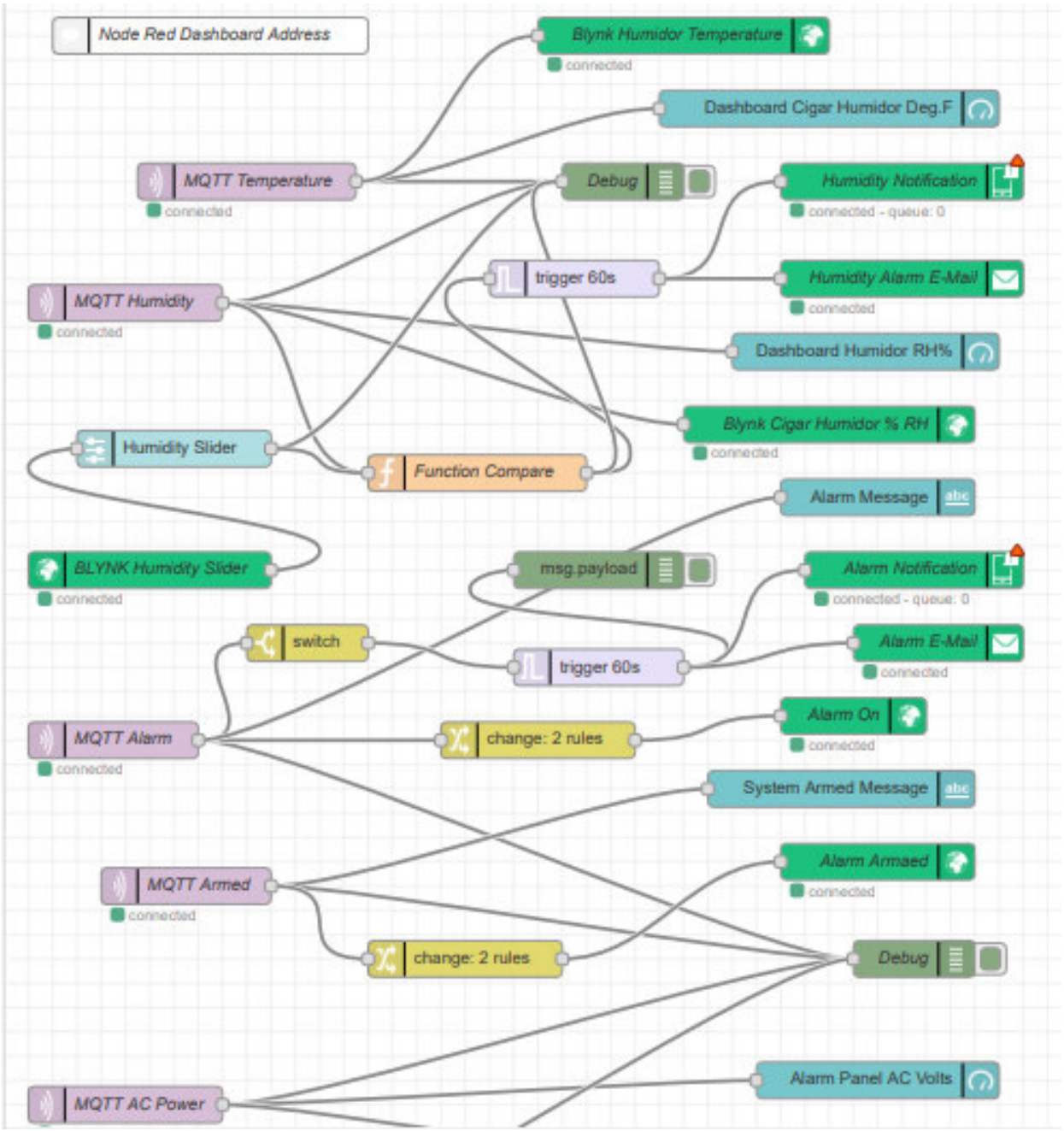
Alarm Horn

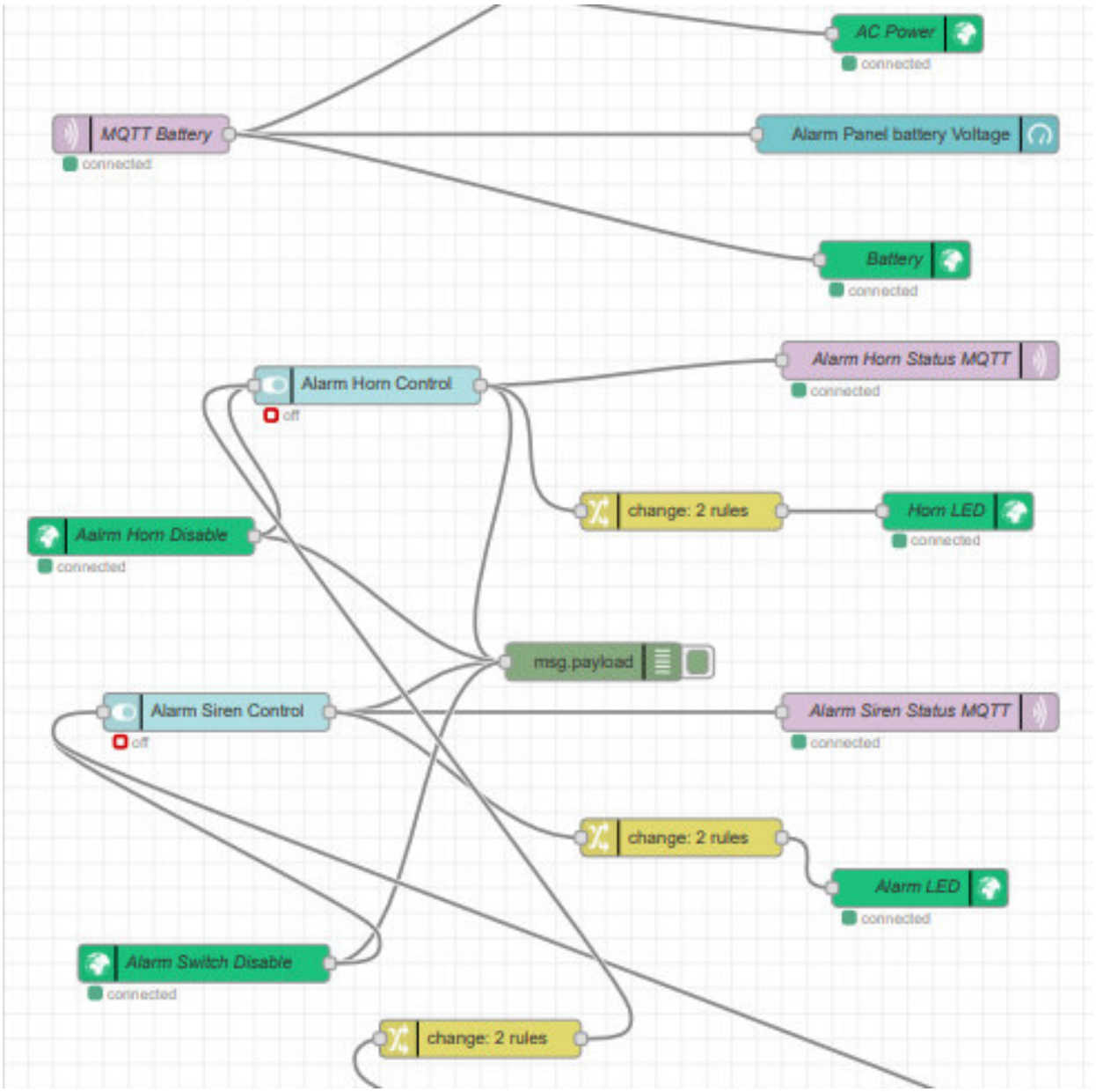
Alarm Siren Switch

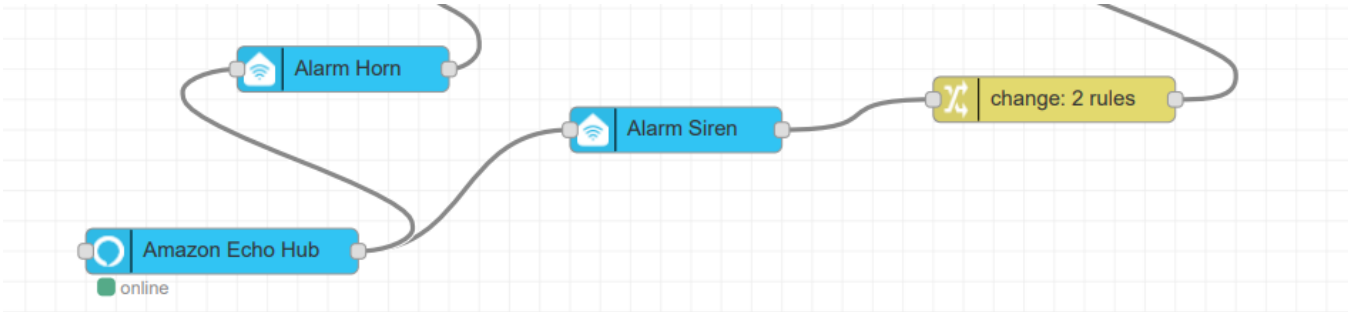
Alarm Siren Control

Humidity Slider

slider 45


Node Red Dashboard Address
Blynk Humidor Temperature
connected
Dashboard Cigar Humidor Deg.F
MQTT Temperature
connected
Debug
Humidity Notification
connected - queue: 0
trigger 60s
Humidity Alarm E-Mail
connected
MQTT Humidity
connected
Dashboard Humidor RH%
Blynk Cigar Humidor % RH
connected
Humidity Slider
Function Compare
Alarm Message
BLYNK Humidity Slider
connected
msg.payload
Alarm Notification
connected - queue: 0
switch
trigger 60s
Alarm E-Mail
connected
MQTT Alarm
connected
change: 2 rules
Alarm On
connected
System Armed Message
MQTT Armed
connected
Alarm Armaed
connected
change: 2 rules
Debug
MQTT AC Power
Alarm Panel AC Volts


AC Power
connected
MQTT Battery
connected
Alarm Panel battery Voltage
Battery
connected
Alarm Horn Control
off
Alarm Horn Status MQTT
connected
change: 2 rules
Horn LED
connected
Aalrm Horn Disable
connected
msg.payload
Alarm Siren Control
off
Alarm Siren Status MQTT
connected
change: 2 rules
Alarm LED
connected
Alarm Switch Disable
connected
change: 2 rules


Alarm Horn
Alarm Siren
change: 2 rules
Amazon Echo Hub
online

ESP8266 MQTT Cigar Humidor Tempaerature and Humidity Publisher
Uses distilled water in a container with a Muffin fan that keeps the Humidity level at 70% using a built in Proportional and Integral Control Algorithm. The fan is PWM controlled by the output of the Controller.

5VDC Muffin Fan

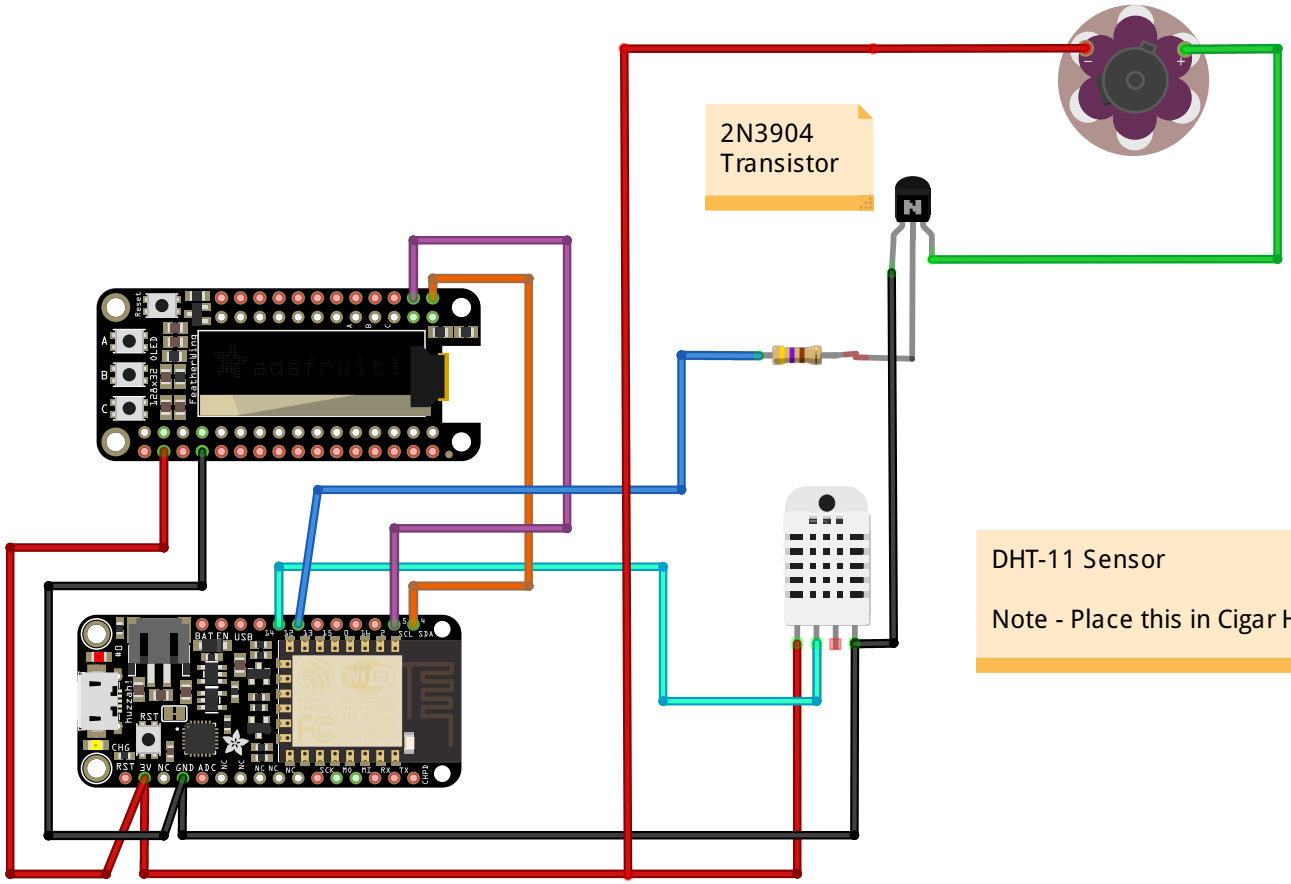
2N3904
Transistor

Switch 'B' Resets Wifi (By deleting 'wifi.dat' File)

Note - Do this if Broker Server IP changes or your router SSID and / or password changes

On-Board 'blue" LED lights up when wifi connection is established.

OLED Display is (128X32)



DHT-11 Sensor

Note - Place this in Cigar Humidor

Bill of Materials: ESP8266 Cigar Humidor.fzz

C:/Fritzing/fritzing.0.9.3b.64.pc/fritzing.0.9.3b.64.pc/sketches/ESP8266 Cigar Humidor.fzz
Sunday, August 18 2019, 10:43:16

Assembly List

Label	Part Type	Properties
DHT1	DHT22 Humidity and Temperature Sensor	
Part1	Adafruit Feather HUZZAH ESP8266	variant variant 1; part # Adafruit #2821
Part2	Adafruit OLED FeatherWing	variant variant 1; part # Adafruit #2900

Shopping List

Amount	Part Type	Properties
1	DHT22 Humidity and Temperature Sensor	
1	Adafruit Feather HUZZAH ESP8266	variant variant 1; part # Adafruit #2821
1	Adafruit OLED FeatherWing	variant variant 1; part # Adafruit #2900

Exported with Fritzing 0.9.3- http://fritzing.org

MQTT Alarm Panel and Raspberry Pi Broker Server

Alarm Pins

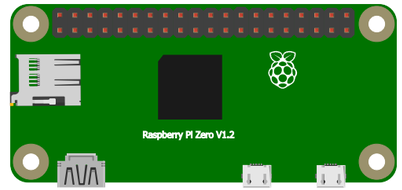
Arm Pins

To Alarm Battery

To Alarm 12VAC

ESP32 Module

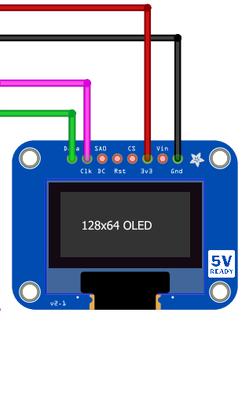
Note - I used a Heltec ESP32 wifi kit available on Amazon (has a buit in 128X64 OLED display). If using Heltec, solder wire on non ground side of 'PGM' button to GPIO23 and use as a wifi Reset to wipe out 'wifi.dat' file.

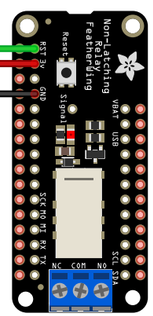


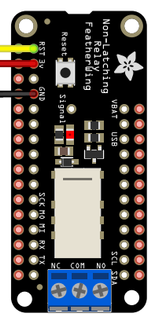
PI Broker Server

Note - Set up per instructions. Also see how to launch 'ui' webpage.

Top USB micro connector requires +5VDC power







Connect Alarm Siren wire (one wire) in series with N/C and Com terminals

Connect Alarm Horn wire (one wire) in series with N/C and Com terminals

Note - I used two 3VDC relays driven with a 2N7000 FET transistor, with 1N914 diode (reverse polarity) across coils to absorp spike.

Bill of Materials: ESP32 and Broker Server.fzz

C:/Fritzing/fritzing.0.9.3b.64.pc/fritzing.0.9.3b.64.pc/sketches/ESP32 and Broker Server.fzz
Sunday, August 18 2019, 15:49:34

Assembly List

Label	Part Type	Properties
C2	Electrolytic Capacitor	capacitance 1000µF; package 1010 [SMD, electrolytic]; voltage 50V
D1	Rectifier Diode	package 300 mil [THT]; type Rectifier; part # 1N4001
D4	Rectifier Diode	package 300 mil [THT]; type Rectifier; part # 1N4001
D5	Rectifier Diode	package 300 mil [THT]; type Rectifier; part # 1N4001
D7	Rectifier Diode	package 300 mil [THT]; type Rectifier; part # 1N4001
JP1	Header 3	variant screw_lock; package screwterminal-3.5mm-3_lock.007s
JP2	Header 3	variant screw_lock; package screwterminal-3.5mm-3_lock.007s
JP3	Header 3	variant screw_lock; package screwterminal-3.5mm-3_lock.007s
JP4	Header 3	variant screw_lock; package screwterminal-3.5mm-3_lock.007s
MCU1	Raspberry Pi Zero	processor Broadcom BCM2835 ARMv11; revision RPI-Zero-V1.1; variant pi zero
Part3	OLED Monochrome 128x64 0.96 inch	variant Mono 128x64 0.96 inch; part # 326
Part5	Adafruit Mini Relay FeatherWing	variant variant 1; part # Adafruit #2895
Part6	Adafruit Mini Relay FeatherWing	variant variant 1; part # Adafruit #2895
R2	22kΩ Resistor	resistance 22kΩ; tolerance ±5%; package 0603 [SMD]
R3	33kΩ Resistor	resistance 33kΩ; tolerance ±5%; package 0603 [SMD]
R4	10kΩ Resistor	resistance 10kΩ; tolerance ±5%; package 0603 [SMD]
R5	10kΩ Resistor	resistance 10kΩ; tolerance ±5%; package 0603 [SMD]
R6	Rotary Potentiometer (Large)	size Rotary - 16mm; package THT; type Rotary Shaft Potentiometer; maximum resistance 10kΩ; track Linear
R7	Rotary Potentiometer (Large)	size Rotary - 16mm; package THT; type Rotary Shaft Potentiometer; maximum resistance 10kΩ; track Linear
R8	22kΩ Resistor	resistance 22kΩ; tolerance ±5%; package 0603 [SMD]
R9	22kΩ Resistor	resistance 22kΩ; tolerance ±5%; package 0603 [SMD]
U1	ESP32-38PinWide	variant Generic; pins 38; type NodeMCU-32S; part # ESP32-38PinWide

Shopping List

Amount	Part Type	Properties
1	Electrolytic Capacitor	capacitance 1000µF; package 1010 [SMD, electrolytic]; voltage 50V
4	Rectifier Diode	package 300 mil [THT]; type Rectifier; part # 1N4001
4	Header 3	variant screw_lock; package screwterminal-3.5mm-3_lock.007s
1	Raspberry Pi Zero	processor Broadcom BCM2835 ARMv11; revision RPI-Zero-V1.1; variant pi zero
1	OLED Monochrome 128x64 0.96 inch	variant Mono 128x64 0.96 inch; part # 326
2	Adafruit Mini Relay FeatherWing	variant variant 1; part # Adafruit #2895
3	22kΩ Resistor	resistance 22kΩ; tolerance ±5%; package 0603 [SMD]
1	33kΩ Resistor	resistance 33kΩ; tolerance ±5%; package 0603 [SMD]
2	10kΩ Resistor	resistance 10kΩ; tolerance ±5%; package 0603 [SMD]
2	Rotary Potentiometer (Large)	size Rotary - 16mm; package THT; type Rotary Shaft Potentiometer; maximum resistance 10kΩ; track Linear
1	ESP32-38PinWide	variant Generic; pins 38; type NodeMCU-32S; part # ESP32-38PinWide

Exported with Fritzing 0.9.3- http://fritzing.org

Setup and Instructions

1) Install Python code files ESP8266 (Cigar Humidor) using "Putty" or "Ampy":
   - The first thing you need to do is download the most recent MicroPython firmware .bin file to load onto your ESP8266 device. You can download it from the MicroPython downloads page.
   - put your device in boot-loader mode, and second you need to copy across the firmware. The exact procedure for these steps is highly dependent on the particular board and you will need to refer to its documentation for details.
   - If you have a board that has a USB connector, a USB-serial convertor, and has the DTR and RTS pins wired in a special way then deploying the firmware should be easy as all steps can be done automatically. Boards that have such features include the Adafruit Feather HUZZAH, Adafruit ESP8266 Breakout, ESP32 boards or NodeMCU boards.
   - For best results it is recommended to first erase the entire flash of your device before putting on new MicroPython firmware.
   - First install Python 3.X. Since my C: Drive is almost full, I installed Python 3.6.3 under D:\Python3.6.3, and placed "Windows Terminal", "Python Idle", and "Python Terminal" on my desktop.
   - Place the micropython firmware version (bin file) in the following path D:\Python3.6.3\Scripts
   - Start windows terminal (normally boots in the C: prompt and type "D:" this will bring you to the D: Prompt. Then type cd D:\Python\3.6.3\Scripts. This will bring you to a new prompt. Now type
   - "pip install esptool", or "pip3 install esptool", depending on the Python version. To ensure no installation errors, open the Python terminal and type "import esptool" and it should take without error. Close the Python Terminal.
   - Note - If you have an old version you almost certainly need to upgrade to the latest version as follows:
   - Pip3 install esptool –upgrade
   - We can now use esptool.py to initially check the connection to the ESP. The quickest way to do this is the command:
   - esptool.py --port COMX flash_id
   - Connect up the Adafruit boards "Feather Huzzah" which windows installs a driver "CP201X" (on COM28) or the Adafruit breakout which requires an FTDI adapter (mine works on COM10). Note- the Adafruit breakout board and some other boards will need to be placed in 'bootloader mode" (see documentation) to erase and flash the ESPXXXX firmware (This is usually a reset or a combination of another button + reset) .
   - Open Windows Terminal and change directory to D:\Python3.6.3\Scripts. Using esptool.py you can erase the ESPXXXX flash by typing the following command:
   - esptool.py --port COMXX erase_flash (where XX is the com port you are using)
   - And then deploy the new firmware by typing the following command for ESP8266:

- esptool.py --port COMXX --baud 460800 write_flash --flash_size=detect 0 esp8266-2017xxxx-vx.xx.bin (where XX is the com port, and xx is the firmware bin version)
- For ESP32 boards use the following command filling in the port and bin number for the ESP download:
- esptool.py --port COM29 --baud 460800 write_flash --flash_size=detect 0x1000 esp32-20180324-v1.9.3-477-g7b0a020a.bin
- You might need to change the "port" setting to something else relevant for your PC. You may also need to reduce the baudrate if you get errors when flashing (eg down to 115200). The filename of the firmware should also match the file that you have.
- For some ESP8266 boards with a particular FlashROM configuration (e.g. some variants of a NodeMCU board) you may need to use the following command to deploy the firmware (note the -fm dio option):
- esptool.py --port COMXX --baud 460800 write_flash --flash_size=detect -fm dio 0 esp8266-2017xxxx-vx.xx.bin
- If the above commands run without error then MicroPython should be installed on your board!
- Using Ampy, install the following files:
    - SSD1306 OLED Library (download Python library from Adafruit on Github)
    - wifimanagerR2 (this is a separate Python file I created to establish a wifi connection) First copy into a Python shell and save file as wifimangerR2.py.
    - ESP82266_main_humidor file (First copy into a Python shell and save file as "main.py" so it will Autoboot on startup.

2) Install Python code files on ESP32 (Alarm Circuit) using "Putty" or "Ampy":
    - Flash micropython on to ESP32 following the steps above for the ESP32 (note- ther eis a different bin file, etc.)
    - Using Ampy, install the following files:
        - SSD1306 OLED Library (download Python library from Adafruit on Github)
        - wifimanagerR1 (this is a separate Python file I created to establish a wifi connection) First copy into a Python shell and save file as wifimangerR1.py.
        - ESP32_main_Alarm file (First copy into a Python shell and save file as "main.py" so it will Autoboot on startup.
3) Set up Raspberry Pi Linux system files by performing the following:

    Set Up Raspberry Pi by performing the following:

    Using NOOBS is the easiest way to install Raspbian on your SD card. To get hold of a copy of NOOBS:

    Visit www.raspberrypi.org/downloads/

    - The simplest option is to download the zip archive of the files.
    - Formatting the SD Card
    - If the SD card on which you wish to install Raspbian currently has an older version of Raspbian on it, you may wish to back up the files from the card first, as they will be overwritten during this process.

- Visit the SD Association's website and download SD Formatter 4.0 for Windows or Mac.
- Follow the instructions to install the software.
- Insert your SD card into the computer or laptop's SD card reader and make a note of the drive letter allocated to it, e.g. F:/.
- In SD Formatter, select the drive letter for your SD card, and format it.
- Extracting NOOBS from the zip archive. Next, you will need to extract the files from the NOOBS zip archive you downloaded from the Raspberry Pi website.
- Go to your Downloads folder and find the zip file you downloaded.
- Extract the files and keep the resulting Explorer/Finder window open.
- Copying the files. Now open another Explorer/Finder window and navigate to the SD card. It's best to position the two windows side by side.
- Select all the files from the NOOBS folder and drag them onto the SD card.
- Once the files have been copied over, insert the micro SD Card into your Raspberry Pi, and plug the Pi into a power source.
- You will be offered a choice when the installer has loaded. You should check the box for Raspbian, and then click Install.
- Click Yes at the warning dialog, and then sit back and relax. It will take a while, but Raspbian will install.
- In a Linux terminal window on the Raspberry Pi type "sudo raspi-config" and enable "VNC" and other options you may want.

Install mosquito Broker Server for MQTT Protocol on Raspberry Pi:

- sudo apt-get upgrade
- sudo apy-get update
- sudo apt install -y mosquitto mosquitto-clients

You'll have to type Y and press Enter to confirm the installation. To make Mosquitto auto start on boot up enter:

- sudo systemctl enable mosquitto.service

Testing Installation

Send the command:

- mosquitto -v

This returns the Mosquitto version that is currently running in your Raspberry Pi. It should be 1.4.X or above.

To use Mosquitto broker later on your projects, you'll need your Raspberry Pi IP address. To retrieve your Raspberry Pi IP address, type the next command in your Terminal window:

- hostname –I

Run Mosquitto on background as a daemon:

- mosquitto -d

Subscribing to testTopic Topic

Advanced Stuff } To subscribe to an MQTT topic with Mosquitto Client open a terminal Window #1 and enter the command:

- mosquitto_sub -d -t testTopic

Publishing “Hello World!” Message to testTopic Topic

To publish a sample message to testTopic, open a terminal Window #2 and run this command:

- mosquitto_pub -d -t testTopic -m "Hello world!"

Note – If you already have an MQTT Publish and / or Subscribe Module working, try the following commands to Test:

a) We can test each "publish topic" by subscribing with the following in a terminal window:

- mosquitto_sub -d -t "/sensor1/temp"(change path for other topics).
- b) We can test each "subscribe topic" by publishing with the following in a terminal window: mosquitto_pub -r -t "/esp8266/4" -m "1" or "0"

4) Set up Node Red by performing the following:
    - Running the following command will download and run the Node Red script.
      bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-deb-package/master/resources/update-nodejs-and-nodered)
    - Note - This script will work on any Debian-based operating system, including Ubuntu and Diet-Pi. You may need to run sudo apt-get install build-essential first to ensure npm is able to build any binary modules it needs to install.
    - If you want Node-RED to run when the Pi is turned on, you can enable the service to autostart by running the command:
      sudo systemctl enable nodered.service
    - To disable the service, run the command:
      sudo systemctl disable nodered.service
    - Once Node-RED is running you can access the editor in a browser. If you are using the browser on the Pi desktop, you can open the address: http://localhost:1880
    - Recommend Chromium or Firefox-ESR on the Pi and not Epiphany
    - Otherwise, you should use the IP address of the Pi: http://<ip-address>:1880 You can find the IP address by running **hostname -I** on the Pi.
    - To install the Node-RED Dashboard run the following command:
      sudo npm install node-red-dashboard
    - The Node-RED application is ready. The dashboard is available at http://<IP address of your Pi>:1880/ui
    - To Install the Alexa library with Node Red perform the following from a Linux Terminal Window:
    - npm install node-red-contrib-amazon-echo
    - Then Add a listening port and reroute PORT 80 to a new Port 8000 by typing the following in a Linux Terminal Window:
      sudo iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT
      sudo iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8000
    - Save the table and have them automatically after reboot:
      sudo apt-get install iptables-persistent.
5) Add the following Blynk Libraries into Node Red by going to the Palette Manager and typing the following under install (may want to use the search window to find)

- node-red-contrib-blynk-ws
- node-red-contrib-blynk-api

6) Import the Node Red Program Nodes & Code by going to the main menu and selecting: Import; then clipboard, and select the "JSON" file enclosed
7) Using your Alexa APP and Account, discover the new devices that were created by the code above to activate them.
8) Connect hardware on ESP8266 in accordance with the schematic
9) Connect hardware on ESP32 in accordance with the schematic
10) Sign up and create a free account with "Blynk" (Visit Website for instructions) and create a mobile app that looks like the picture enclosed. Special settings within the App are as follows:
    - Choose Raspberry Pi B+ as the device
    - For gauges the set the min and max range and the significant digits are set by placing /pin.#/ in the label field for one decimal place.
    - Cigar Humidor Temp gauge = V1, range is 0-120
    - Cigar Humidor Humidity gauge = V0, range is 0-100
    - Cigar Humidor Slider = V10, range is 0-100
    - Alarm battery = V3, range is 0-14
    - Alarm Ac Power = V2, range is 0-140
    - Add Notification Block
    - Add E-Mail Block
    - Alarm Siren Disable = V5
    - Alarm Horn Disable = V4
    - Armed LED = V9
    - Alarm LED = V8
    - Siren LED = V7
    - Horn LED = V6
    - Choose any color you want for each widget

```
# Import Modules
# --------------
import network, time
from time import sleep
import wifimanagerR2 # Start the Wifi Manager Program (this is a seperate file)
import os, network, machine
from ubinascii import hexlify
from machine import unique_id
from umqtt.simple import MQTTClient
from network import WLAN
from machine import Pin, I2C, PWM
from ssd1306 import SSD1306_I2C
#from wifimanagerR2 import wifimanagerR2
from dht import DHT22 # Change to DHT11 for the 11 type sensor

# ESP8266 GPIO Pins:
# ------------------
WIFI_LED_PIN = 2 # Wifi Status Pin
RST_PIN = 16 # Reset WiFI and Reboot
DHT_PIN = 12

# PI Control Initial Values
# -------------------------
output = 0.0
delta_time = 0.05
I_Term = 0.0
feedback = 0.0 # DHT Reading
frequency = 1000 # Range is 1Hz to 1 KHz

# Hardare ESP8266 Setup:
# ----------------------
i2c = I2C(-1, sda = Pin(4), scl = Pin(5)) # Set Up OLED I2C Pins with software mode
display = SSD1306_I2C(128, 32, i2c) # I2C parameters for OLED Display
RESET = (Pin(RST_PIN, Pin.IN))# On board pull-up resistor enabled
d = DHT22(Pin(DHT_PIN, Pin.IN, Pin.PULL_UP))# Change to DHT11 for the 11 type sensor
blueled = Pin(WIFI_LED_PIN, Pin.OUT, value=1) # Sets up wifi LED and starts off
fan = PWM(Pin(13), frequency) # PWM object on pin 5, at specified frequency

# Global Variables & Constants:
# -----------------------------
MESUREMENT_INTERVAL = 5 # 5 Seconds
last_mesurement_time = 0
SERVER = ""  # Rasperry pi IP Address
CLIENT_ID = hexlify(unique_id())
TOPIC1 = b"/sensor1/temp"
TOPIC2 = b"/sensor1/hum"
profiles = {} # Dictionary to hold keys & information

wlan = wifimanagerR2.get_connection()
if wlan is None:
    print("Could Not Initialize Network Connection.")
    L1 = "Could Not"
    L2 = "Initialize"
    L3 = "WiFi Network"
    delayx = 3
    Display_Stat(L1,L2,L3,delayx) # Goto Display function
    while True:  # Program will not go further and stat_mode1() will not operate unless
    line is commented out.
        pass

# Shut off Access Point
# ---------------------
ap_if = network.WLAN(network.AP_IF)
sleep(5) # 5 second time delay
ap_if.active(False) # Turn off AP Mode
```

```
67  # Generic Function to Display 3 lines on OLED
68  # ----------------------------------------
69  def Display_Stat(line1,line2,line3,delaytime):
70      display.fill(0) # Clear Previous Text
71      display.text(line1,0,0,1) #Line 1
72      display.text(line2,0,10,1) # Line 2
73      display.text(line3,0,20,1) # Line 3
74      display.show() # Display New Text
75      sleep(delaytime) # Delay secs
76
77  # Check Wifi Data File for Broker IP Address
78  # ----------------------------------------
79  try:
80      profiles = wifimanagerR2.read_profiles()
81      for line in profiles:
82          SERVER = profiles['IP']
83      print("Humidor Read Profiles = " + str(profiles)) # Debug
84      print("Humidor Broker Server IP : " + SERVER)
85  except Exception as e:
86       print(e)
87       L1 = "Cannot Find"
88       L2 = "Wifi Data File"
89       L3 = "Trouble Shoot"
90       delayx = 3
91       Display_Stat(L1,L2,L3,delayx) # Goto function
92       sta_if = network.WLAN(network.STA_IF)
93       sta_if.active(True)
94       sta_if.connect('ssidhome', 'password')
95       #wifimanagerR2.do_connect('ssid', 'password') # Goto wifi program function
96       #wifimanagerR2.get_connection() # Goto wifi program function
97
98
99  # Function Check to see if Station mode wifi is active
100 # ----------------------------------------------------
101 def stat_mode1():
102     sta1 = WLAN(network.STA_IF)
103     if not sta1.isconnected():
104         blueled.value(1) # Turn off LED
105         # Display Message:
106         L1 = "Press Reset"
107         L2 = "To Re-Start"
108         L3 = "WiFi Connection"
109         delayx = 2
110         Display_Stat(L1,L2,L3,delayx) # Goto function
111         #while True:  # Program will not go further and stat_mode1() will not operate
            unless line is commented out.
112             #pass
113     else: # connected to station wifi
114         blueled.value(0) # Turn on LED
115
116 # Function to Read Temp and Humidity
117 # ----------------------------------
118 def TemHum():
119     try:
120         d.measure()
121         t = d.temperature()
122         tf = "{0:.2f}".format((t*(9/5.0))+32) # Convert to a unit and a string
123         h = "{0:.1f}".format(d.humidity()) # Convert to a unit and a string
124         print("Temperature Deg.F. =: " + tf)
125         print("% Humidity =: " + h)
126         return (tf,h)
127     except Exception as e:
128         print(e)
129         return (0,0)
130
131 # PI Controller Function
132 # ----------------------
```

```
def PI (Kp=78.6, Ki=16.5, Kd=0.0, setpoint = 70):  # Initialize Gains & Setpoint
    global output, I_Term, feedback
    sleep(delta_time) # Controller sample rate
    int_error = 0.0
    windup_guard = 25.0
    error = setpoint - feedback  # Error Term
    I_Term += error * delta_time # Intergral Term
    if (I_Term > windup_guard): # Positive Integral Windup Guard
        I_Term = windup_guard
    elif (I_Term < -windup_guard): # Negative Integral Windup Guard
        I_Term = -windup_guard
    print("I_Term = " + str(I_Term)) # Debug
    print("P_Term = " + str(Kp * error)) # Debug
    output = (Kp * error) + (Ki * I_Term) # Controller Output (Proportional + Integral)
    print("Output = " + str(output)) # Debug
    if output <= 0: # Limit output to positive PWM values only
        output = 0
    if output >= 1023: # Limit output to positive PWM values only
        output = 1023
    return (output)

# Function to Publish MQTT Readings:
# ----------------------------------
def envioMQTT(server=SERVER, topic="/stuff", data=None):
    try:
        c = MQTTClient(CLIENT_ID, server)
        c.connect()
        c.publish(topic, data)
        sleep(0.2)
        c.disconnect()
    except Exception as e:
        print(e)
        pass

# Function to Reboot:
# -------------------
def reboot():
    print("Re-Booting......")
    L1 = "Re-Booting......"
    L2 = ""
    L3 = ""
    delayx = 3
    Display_Stat(L1,L2,L3,delayx) # Goto function
    machine.reset() # Perform a hardware reset

# Start of Main Program:
# ======================

# Main Banner
# -----------
Lx = "ESP8266 Humidor"
Ly = "  Created By:"
Lz = "Roy H Guerra Jr."
delayxx = 3
Display_Stat(Lx,Ly,Lz,delayxx) # Goto function

while True:
    try:
        stat_mode1()# Goto Function
        if RESET.value() == 0:
            os.remove("wifi.dat")
            reboot() # Goto Function

        current_time = time.time() # Get a time stamp

        if current_time - last_mesurement_time > MESUREMENT_INTERVAL:
            (tf,h) = TemHum() # Goto get temp & hum Function
```

```
            feedback = float(h)
            print("Feedback = " + str(feedback))
            PI_Out = int(PI()) # Goto Function and convert to integer
            print("PWM = " + str(PI_Out)) # Debug
            fan.duty(PI_Out) # Fan Duty-Cycle from Controller Output
            # Place OLED display function here
            envioMQTT(SERVER,TOPIC1,tf) #Publish Topic
            envioMQTT(SERVER,TOPIC2,h) #Publish Topic
            # Display Message:
            L1 = "Conditions Are:"
            L2 = "  Temp: " + tf + "_F"
            L3 = "  Hum: " + h + "%"
            delayx = 0.1
            Display_Stat(L1,L2,L3,delayx) # Goto function
            last_mesurement_time = current_time # New time stamp
    except Exception as e:
                    print(e)
             # Main Selection Screen Page Stuff
                    L1 = "Bad Hardware"
                    L2 = "and / or a"
                    L3 = "Program Issue"
                    delayx = 2
                    Display_Stat(L1,L2,L3,delayx) # Goto Display function
                    #pass

```

```
# Import Modules:
# ---------------
from time import sleep
import time, os, network, machine
from ubinascii import hexlify
from machine import unique_id
from umqtt.simple import MQTTClient
from network import WLAN
from machine import Pin, I2C, ADC
from ssd1306 import SSD1306_I2C
import wifimanagerR1

# ESP32 GPIO Pins:
# ----------------
WIFI_LED_PIN = 25 # Onboard white LED
OLED_RST_PIN = 16 # Onboard OLED Reset Pin
Batt_AD_PIN = 36 # Scaled Voltage to read input
PWR_AD_PIN = 37 # Scaled Voltage to read input
ALARM_PIN = 38 # Scaled Digital Input from 12VDC to 3.3VDc
ARM_PIN = 21 # Scaled Digital Input from 12VDC to 3.3VDc
Horn_PIN = 22 # Digital Output through Relay to cut Off
Siren_PIN = 19 # Digital Output through Relay to cut Off
RST_PIN = 23 # Reset WiFI and Reboot

# Hardare ESP32 Setup:
# --------------------
led1 = Pin(WIFI_LED_PIN, Pin.OUT, value=0) # Sets up WiFi LED to Low
OLED_RST = Pin(OLED_RST_PIN, Pin.OUT, value=1) # Sets up OLED RST to high
i2c = I2C(sda = Pin(4), scl = Pin(15)) # Set Up OLED IIC Pins
display = SSD1306_I2C(128, 64, i2c) # IIC parameters for OLED Display
Horn_Off = Pin(Horn_PIN, Pin.OUT, value=0) # Output pin, start off
Siren_Off = Pin(Siren_PIN, Pin.OUT, value=0) # Output pin, start off
SYS_ARM = Pin(ARM_PIN, Pin.IN, Pin.PULL_UP)# pull-up resistor enabled
SYS_ALARM = Pin(ALARM_PIN, Pin.IN, Pin.PULL_UP)# pull-up resistor enabled
RESET = Pin(RST_PIN, Pin.IN, Pin.PULL_UP)# pull-up resistor enabled
Bat_Level = ADC(Pin(Batt_AD_PIN)) # A/D Class Object
Bat_Level.atten(ADC.ATTN_11DB) # 11 dB attenuation means full 0 - 3.3V range
Power_Level = ADC(Pin(PWR_AD_PIN)) # A/D Class Object
Power_Level.atten(ADC.ATTN_11DB) # 11 dB attenuation means full 0 - 3.3V range

# Global Variables & Constants:
# -----------------------------
MESUREMENT_INTERVAL = 6 # 6 Seconds
last_mesurement_time = 0
x = "" # String storage variable
y = "" # String storage variable
SERVER = ""  # Rasperry pi IP Address
CLIENT_ID = hexlify(unique_id())
TOPIC1 = b"/sensor1/batt"
TOPIC2 = b"/sensor2/power"
TOPIC3 = b"/sensor3/Armed"
TOPIC4 = b"/sensor4/Alarm"
TOPIC5 = b"/esp32/22"
TOPIC6 = b"/esp32/19"
profiles = {} # Dictionary to hold keys & information

# Establish Internet Connection:
# ------------------------------
wlan = wifimanagerR1.get_connection()
if wlan is None:
    print("Could Not Initialize Network Connection.")
    L1 = "Could Not"
    L2 = "Initialize"
    L3 = "Network"
    L4 = "Connection"
    L5 = "Please Turn Off"
    L6 = "And Try Again"
```

```
    delayx = 3
    Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
    #while True:  # Program will not go further and stat_mode1() will not operate unless line is commented out.
        #pass

# Shut off Access Point
# ---------------------
ap_if = network.WLAN(network.AP_IF)
sleep(5) # 5 second time delay
ap_if.active(False) # Turn off AP Mode

# Generic Function to Display 6 lines on OLED
# -------------------------------------------
def Display_Stat1(line1,line2,line3,line4,line5,line6,delaytime):
    display.fill(0) # Clear Previous Text
    display.text(line1,0,0,1) #Line 1
    display.text(line2,0,10,1) # Line 2
    display.text(line3,0,20,1) # Line 3
    display.text(line4,0,30,1) #Line 4
    display.text(line5,0,40,1) # Line 5
    display.text(line6,0,50,1) # Line 6
    display.show() # Display New Text
    sleep(delaytime) # Delay secs

# Check Wifi Data File for Broker IP Address
# ------------------------------------------
try:
    profiles = wifimanagerR1.read_profiles()
    for line in profiles:
        SERVER = profiles['IP']
    print("Alarm Panel Read Profiles = " + str(profiles)) # Debug
    print("Alarm Panel Broker Server IP : " + SERVER)
except Exception as e:
     print(e)
     L1 = "Cannot Find"
     L2 = "Wifi Data File"
     L3 = ""
     L4 = "Press Reset"
     L5 = "Button To Start"
     L6 = "Wifi Server...."
     delayx = 3
     Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto function
     #os.remove('wifi.dat') # Clear the wifi settings file
     #reboot() # Goto Function
     #sta_if = network.WLAN(network.STA_IF) # Only for ESP8266
     #sta_if.active(True) # Only for ESP8266
     #sta_if.connect('ssidhome', 'password') # Only for ESP8266
     #wifimanagerR2.do_connect('ssid', 'password') # Goto wifi program function
     #wifimanagerR2.get_connection() # Goto wifi program function

# Function Check to see if Station mode wifi is active
# ----------------------------------------------------
def stat_mode1():
    sta1 = WLAN(network.STA_IF)
    if not sta1.isconnected():
        led1.value(0) # Turn off LED
        # Display Message:
        L1 = "Press Reset"
        L2 = "To Re-Start"
        L3 = "WiFi Connection"
        L4 = "or"
        L5 = "Check Router"
        L6 = "Control Settings"
        delayx = 2
        Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto function
    else: # connected to station wifi
```

```
134         led1.value(1) # Turn on LED
135
136 # Function to Publish MQTT Readings:
137 # ---------------------------------
138 def envioMQTT(server=SERVER, topic="/stuff", data=None):
139     try:
140         c = MQTTClient(CLIENT_ID, server)
141         c.connect()
142         c.publish(topic, data)
143         sleep(0.2)
144         c.disconnect()
145     except Exception as e:
146         print(e)
147         pass
148
149 # Function to Reboot:
150 # -------------------
151 def reboot():
152     print("Re-Booting.....")
153     L1 = "Re-Booting....."
154     L2 = ""
155     L3 = ""
156     L4 = ""
157     L5 = ""
158     L6 = ""
159     delayx = 3
160     Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto function
161     machine.reset() # Perform a hardware reset
162
163 # Function to Parse MQTT Subscription Topic:
164 # ------------------------------------------
165 def sub_cb(topic, msg):
166     print((topic, msg))
167     if topic == b"/esp32/22":
168         if msg == b"1":
169             Horn_Off.value(1) # Horn is activated Off
170         elif msg == b"0":
171             Horn_Off.value(0) # Horn is normally connected
172     if topic == b"/esp32/19":
173         if msg == b"1":
174             Siren_Off.value(1) # Siren is activated Off
175         elif msg == b"0":
176             Siren_Off.value(0) # Siren is normally connected
177
178 # Function to Subscribe to MQTT Topic:
179 # ------------------------------------
180 def recepcionMQTT(server, topic):
181     c = MQTTClient(CLIENT_ID, server)
182     # Subscribed messages will be delivered to this callback
183     c.set_callback(sub_cb)
184     c.connect()
185     c.subscribe(topic)
186     print("Connected to %s, subscribed to %s topic" % (server, topic))
187     try:
188         c.wait_msg()
189     finally:
190         c.disconnect()
191
192 # Function to Convert Voltages on Alarm Panel:
193 # --------------------------------------------
194 def Volts_Conversion():
195     global Bat_Level, Power_Level
196     Num1 = 0
197     Num2 = 0
198     Num1 = Bat_Level.read() # Read A/D
199     Scaled_Bat_V = "{0:.1f}".format(Num1 * (12.0 / 4095)) # Convert to a scaled unit
        and a string
```

```
200         print("Battery Voltage = " + Scaled_Bat_V) # Debug
201         Num2 = Power_Level.read() # Read A/D
202         Scaled_AC_V = "{0:.1f}".format(Num2 * (120.0 / 4095)) # Convert to a scaled unit
            and a string
203         print("AC Power In = " + Scaled_AC_V) # Debug
204         return (Scaled_Bat_V, Scaled_AC_V)
205
206     # Start of Main Program:
207     # ======================
208
209     # Main Banner
210     # -----------
211     L1 = "MQTT Pub & Sus"
212     L2 = "For Alalrm"
213     L3 = "Panel Monitor"
214     L4 = "To Broker Server"
215     L5 = "Created By:"
216     L6 = "Roy H Guerra Jr."
217     delayx = 3
218     Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto function
219
220     while True:
221         try:
222             stat_mode1()# Goto Function
223             if RESET.value() == 0:
224                 os.remove('wifi.dat')
225                 reboot() # Goto Function
226
227             current_time = time.time() # Get a time stamp
228             recepcionMQTT(SERVER,TOPIC5) #Subscription Topic
229             sleep(0.1)
230             recepcionMQTT(SERVER,TOPIC6) # Subscription Topic
231             sleep(0.1)
232
233             if current_time - last_mesurement_time > MESUREMENT_INTERVAL:
234                 (Scaled_Bat_V, Scaled_AC_V) = Volts_Conversion() # Goto
                    Function
235                 envioMQTT(SERVER,TOPIC1,Scaled_Bat_V) #Publish Topic
236                 envioMQTT(SERVER,TOPIC2,Scaled_AC_V) #Publish Topic
237                 if SYS_ARM.value() == 1:
238                     x = "On"
239                     envioMQTT(SERVER,TOPIC3,x) #Publish Topic
240                 elif SYS_ARM.value() == 0:
241                     x = "Off"
242                     envioMQTT(SERVER,TOPIC3,x) #Publish Topic
243                 if SYS_ALARM.value() == 1:
244                     y = "On"
245                     envioMQTT(SERVER,TOPIC4,y) #Publish Topic
246                 elif SYS_ALARM.value() == 0:
247                     y = "Off"
248                     envioMQTT(SERVER,TOPIC4,y) #Publish Topic
249                 # Display Messages:
250                 # -----------------
251                 L1 = "  Alarm Panel  "
252                 L2 = "---------------"
253                 L3 = "Batt_V = " + Scaled_Bat_V
254                 L4 = "AC_V IN = " + Scaled_AC_V
255                 L5 = "Arm Loop = " + x
256                 L6 = "Alarm is = " + y
257                 delayx = 0.2
258                 Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto function
259                 last_mesurement_time = current_time # New time stamp
260         except Exception as e:
261                         print(e)
262                         # Main Selection Screen Page Stuff
263                         L1 = "Bad Hardware"
264                         L2 = ""
```

```
                    L3 = "and / or a"
                    L4 = ""
                    L5 = "Program Issue"
                    L6 = ""
                    delayx = 2
                    Display_Stat1(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
                    #pass
```

```
[{"id":"9c3f43b0.52eba","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT
Temperature","topic":"/sensor1/temp","qos":"0","datatype":"auto","broker"
:"6fba49cc.7e3018","x":200,"y":160,"wires":[["ad1ba6f6.acbaa8","2a8aafef.
2f198","31ea2dba.26e352"]]},{"id":"ad1ba6f6.acbaa8","type":"debug","z":"5
d0da482.f4417c","name":"Debug","active":true,"tosidebar":true,"console":f
alse,"tostatus":false,"complete":"payload","targetType":"msg","x":510,"y"
:160,"wires":[]},{"id":"f980805f.47f78","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT
Humidity","topic":"/sensor1/hum","qos":"0","datatype":"auto","broker":"6f
ba49cc.7e3018","x":100,"y":260,"wires":[["ad1ba6f6.acbaa8","6c2bd18d.e880
f","36ae3ad1.35a856","6c9e341.c5122cc"]]},{"id":"2a8aafef.2f198","type":"
ui_gauge","z":"5d0da482.f4417c","name":"Dashboard Cigar Humidor
Deg.F","group":"f671b92d.bf2c58","order":0,"width":0,"height":0,"gtype":"
gage","title":"Cigar Humidor Temperature","label":"Deg.
F","format":"{{value}}","min":0,"max":"120","colors":["#00b500","#e6e600"
,"#ca3838"],"seg1":"","seg2":"","x":680,"y":100,"wires":[]},{"id":"6c2bd1
8d.e880f","type":"ui_gauge","z":"5d0da482.f4417c","name":"Dashboard
Humidor
RH%","group":"9f88262d.a77a28","order":0,"width":0,"height":0,"gtype":"ga
ge","title":"Cigar Humidor
Humidity","label":"%RH","format":"{{value}}","min":0,"max":"100","colors"
:["#00b500","#e6e600","#ca3838"],"seg1":"","seg2":"","x":710,"y":300,"wir
es":[]},{"id":"d90ce77d.fcb4d8","type":"comment","z":"5d0da482.f4417c","n
ame":"Node Red Dashboard Address","info":"The dashboard is available
at\nhttp://<IP address of your
Pi>:1880/ui","x":170,"y":40,"wires":[]},{"id":"31ea2dba.26e352","type":"b
lynk-api-out-write","z":"5d0da482.f4417c","name":"Blynk Humidor
Temperature","type_connect":"api","pin_type":"virtual","pin":0,"client":"
f9c90265.5b199","x":560,"y":40,"wires":[]},{"id":"36ae3ad1.35a856","type"
:"blynk-api-out-write","z":"5d0da482.f4417c","name":"Blynk Cigar Humidor
%
RH","type_connect":"api","pin_type":"virtual","pin":"1","client":"f9c9026
5.5b199","x":680,"y":360,"wires":[]},{"id":"9dc7925f.0c7b4","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT
Alarm","topic":"/sensor4/Alarm","qos":"0","datatype":"auto","broker":"6fb
a49cc.7e3018","x":90,"y":620,"wires":[["cebcf37b.746e6","1556a29d.5a985d"
,"90f5621c.84b72","2f9c5379.da9b4c"]]},{"id":"cebcf37b.746e6","type":"deb
ug","z":"5d0da482.f4417c","name":"Debug","active":true,"tosidebar":true,"
console":false,"tostatus":false,"complete":"payload","targetType":"msg","
x":750,"y":800,"wires":[]},{"id":"20d7181e.ee2f88","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT
Armed","topic":"/sensor3/Armed","qos":"0","datatype":"auto","broker":"6fb
a49cc.7e3018","x":150,"y":740,"wires":[["cebcf37b.746e6","a65de65a.8e13b8
","c59f3ffe.2ae6c"]]},{"id":"2832dd31.8bece2","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT AC
Power","topic":"/sensor2/power","qos":"0","datatype":"auto","broker":"6fb
a49cc.7e3018","x":100,"y":920,"wires":[["cebcf37b.746e6","42fd5d97.691f34
","abcf8643.8e8d98"]]},{"id":"42a9f75f.0d9d58","type":"mqtt
in","z":"5d0da482.f4417c","name":"MQTT
Battery","topic":"/sensor1/batt","qos":"0","datatype":"auto","broker":"6f
ba49cc.7e3018","x":110,"y":1060,"wires":[["cebcf37b.746e6","6320933f.c0d0
6c","8647ba41.ee2cb8"]]},{"id":"6320933f.c0d06c","type":"ui_gauge","z":"5
d0da482.f4417c","name":"Alarm Panel battery
```

```
Voltage","group":"8711b80a.b96c48","order":0,"width":0,"height":0,"gtype"
:"gage","title":"Alarm Panel Battery Voltage","label":"Volts
DC","format":"{{value}}","min":0,"max":"14","colors":["#00b500","#e6e600"
,"#ca3838"],"seg1":"","seg2":"","x":720,"y":1060,"wires":[]},{"id":"42fd5
d97.691f34","type":"ui_gauge","z":"5d0da482.f4417c","name":"Alarm Panel
AC
Volts","group":"f3d16070.eff6f","order":0,"width":0,"height":0,"gtype":"g
age","title":"Alarm Panel AC Volts","label":"Volts
AC","format":"{{value}}","min":0,"max":"140","colors":["#00b500","#e6e600
","#ca3838"],"seg1":"","seg2":"","x":720,"y":900,"wires":[]},{"id":"2f9c5
379.da9b4c","type":"ui_text","z":"5d0da482.f4417c","group":"f22674fd.edef
c8","order":0,"width":0,"height":0,"name":"Alarm Message","label":"Alarm
Message","format":"{{msg.payload}}","layout":"row-
spread","x":720,"y":420,"wires":[]},{"id":"a65de65a.8e13b8","type":"ui_te
xt","z":"5d0da482.f4417c","group":"ae170b9.f25eef8","order":0,"width":0,"
height":0,"name":"System Armed Message","label":"System Armed
Message","format":"{{msg.payload}}","layout":"row-
spread","x":690,"y":660,"wires":[]},{"id":"52d1f801.3f2258","type":"mqtt
out","z":"5d0da482.f4417c","name":"Alarm Horn Status
MQTT","topic":"/esp32/22","qos":"2","retain":"true","broker":"6fba49cc.7e
3018","x":730,"y":1240,"wires":[]},{"id":"4f63a64a.db0b78","type":"mqtt
out","z":"5d0da482.f4417c","name":"Alarm Siren Status
MQTT","topic":"/esp32/19","qos":"2","retain":"true","broker":"6fba49cc.7e
3018","x":730,"y":1520,"wires":[]},{"id":"96599bd2.82aeb8","type":"debug"
,"z":"5d0da482.f4417c","name":"","active":true,"tosidebar":true,"console"
:false,"tostatus":false,"complete":"false","x":470,"y":1480,"wires":[]},{
"id":"25062ef6.6c2e02","type":"ui_switch","z":"5d0da482.f4417c","name":"A
larm Horn Control","label":"Alarm
Horn","tooltip":"","group":"2edc2351.f8cb4c","order":0,"width":0,"height"
:0,"passthru":true,"decouple":"false","topic":"/esp32/22","style":"","onv
alue":"1","onvalueType":"str","onicon":"","oncolor":"","offvalue":"0","of
fvalueType":"str","officon":"","offcolor":"","x":290,"y":1260,"wires":[["
96599bd2.82aeb8","52d1f801.3f2258","5b601705.028928"]]},{"id":"348434fb.1
a577c","type":"ui_switch","z":"5d0da482.f4417c","name":"Alarm Siren
Control","label":"Alarm Siren
Control","tooltip":"","group":"6689f280.8135ac","order":0,"width":0,"heig
ht":0,"passthru":true,"decouple":"false","topic":"/esp32/19","style":"","
onvalue":"1","onvalueType":"str","onicon":"","oncolor":"","offvalue":"0",
"offvalueType":"str","officon":"","offcolor":"","x":170,"y":1520,"wires":
[["96599bd2.82aeb8","4f63a64a.db0b78","5595e062.984d7"]]},{"id":"8647ba41
.ee2cb8","type":"blynk-api-out-
write","z":"5d0da482.f4417c","name":"Battery","type_connect":"api","pin_t
ype":"virtual","pin":"3","client":"f9c90265.5b199","x":710,"y":1160,"wire
s":[]},{"id":"abcf8643.8e8d98","type":"blynk-api-out-
write","z":"5d0da482.f4417c","name":"AC
Power","type_connect":"api","pin_type":"virtual","pin":"2","client":"f9c9
0265.5b199","x":720,"y":980,"wires":[]},{"id":"18bd518.7bc93af","type":"b
lynk-api-in-write","z":"5d0da482.f4417c","name":"Aalrm Horn
Disable","pin":"4","pin_type":"virtual","pin_all":0,"client":"f9c90265.5b
199","x":110,"y":1380,"wires":[["96599bd2.82aeb8","25062ef6.6c2e02"]]},{"
id":"bfc33c15.51bfd","type":"blynk-api-in-
write","z":"5d0da482.f4417c","name":"Alarm Switch
Disable","pin":"5","pin_type":"virtual","pin_all":0,"client":"f9c90265.5b
199","x":160,"y":1720,"wires":[["96599bd2.82aeb8","348434fb.1a577c"]]},{"
```

```
id":"a3f44249.4d086","type":"blynk-api-out-
write","z":"5d0da482.f4417c","name":"Horn
LED","type_connect":"api","pin_type":"virtual","pin":"6","client":"f9c902
65.5b199","x":760,"y":1360,"wires":[]},{"id":"bb721d50.d23e","type":"blyn
k-api-out-write","z":"5d0da482.f4417c","name":"Alarm
LED","type_connect":"api","pin_type":"virtual","pin":"7","client":"f9c902
65.5b199","x":730,"y":1660,"wires":[]},{"id":"5b601705.028928","type":"ch
ange","z":"5d0da482.f4417c","name":"","rules":[{"t":"change","p":"payload
","pt":"msg","from":"1","fromt":"str","to":"0","tot":"num"},{"t":"change"
,"p":"payload","pt":"msg","from":"0","fromt":"str","to":"1024","tot":"num
"}],"action":"","property":"","from":"","to":"","reg":false,"x":540,"y":1
360,"wires":[["a3f44249.4d086"]]},{"id":"5595e062.984d7","type":"change",
"z":"5d0da482.f4417c","name":"","rules":[{"t":"change","p":"payload","pt"
:"msg","from":"1","fromt":"str","to":"0","tot":"num"},{"t":"change","p":"
payload","pt":"msg","from":"0","fromt":"str","to":"1024","tot":"num"}],"a
ction":"","property":"","from":"","to":"","reg":false,"x":540,"y":1620,"w
ires":[["bb721d50.d23e"]]},{"id":"5e0d2ecd.2b8a5","type":"blynk-api-out-
write","z":"5d0da482.f4417c","name":"Alarm
On","type_connect":"api","pin_type":"virtual","pin":"8","client":"f9c9026
5.5b199","x":700,"y":600,"wires":[]},{"id":"c42f8026.8b919","type":"blynk
-api-out-write","z":"5d0da482.f4417c","name":"Alarm
Armaed","type_connect":"api","pin_type":"virtual","pin":"9","client":"f9c
90265.5b199","x":720,"y":720,"wires":[]},{"id":"1556a29d.5a985d","type":"
change","z":"5d0da482.f4417c","name":"","rules":[{"t":"change","p":"paylo
ad","pt":"msg","from":"On","fromt":"str","to":"1024","tot":"num"},{"t":"c
hange","p":"payload","pt":"msg","from":"Off","fromt":"str","to":"0","tot"
:"num"}],"action":"","property":"","from":"","to":"","reg":false,"x":440,
"y":620,"wires":[["5e0d2ecd.2b8a5"]]},{"id":"c59f3ffe.2ae6c","type":"chan
ge","z":"5d0da482.f4417c","name":"","rules":[{"t":"change","p":"payload",
"pt":"msg","from":"On","fromt":"str","to":"1024","tot":"num"},{"t":"chang
e","p":"payload","pt":"msg","from":"Off","fromt":"str","to":"o","tot":"nu
m"}],"action":"","property":"","from":"","to":"","reg":false,"x":380,"y":
800,"wires":[["c42f8026.8b919"]]},{"id":"df04863e.7746a8","type":"debug",
"z":"5d0da482.f4417c","name":"","active":true,"tosidebar":true,"console":
false,"tostatus":false,"complete":"false","x":490,"y":480,"wires":[]},{"i
d":"83ce9832.e481d8","type":"blynk-api-in-
write","z":"5d0da482.f4417c","name":"BLYNK Humidity
Slider","pin":"10","pin_type":"virtual","pin_all":0,"client":"f9c90265.5b
199","x":120,"y":480,"wires":[["9a9545aa.c4fcf8"]]},{"id":"9a9545aa.c4fcf
8","type":"ui_slider","z":"5d0da482.f4417c","name":"Humidity
Slider","label":"slider","tooltip":"","group":"a2fcc157.893b8","order":0,
"width":0,"height":0,"passthru":true,"outs":"end","topic":"Humidity
Slider","min":0,"max":"100","step":1,"x":140,"y":380,"wires":[["ad1ba6f6.
acbaa8","6c9e341.c5122cc"]]},{"id":"66e4c5c0.21ffac","type":"blynk-ws-
out-notify","z":"5d0da482.f4417c","name":"Alarm
Notification","client":"55599e05.7bf55","queue":true,"rate":"10","x":750,
"y":480,"wires":[]},{"id":"90f5621c.84b72","type":"switch","z":"5d0da482.
f4417c","name":"","property":"payload","propertyType":"msg","rules":[{"t"
:"eq","v":"On","vt":"str"}],"checkall":"true","repair":false,"outputs":1,
"x":250,"y":540,"wires":[["ee984a2e.cd3548"]]},{"id":"99a8cbe9.315ae8","t
ype":"blynk-api-out-email","z":"5d0da482.f4417c","name":"Alarm E-
Mail","email":"u003rhg@gmail.com","client":"f9c90265.5b199","x":770,"y":5
40,"wires":[]},{"id":"ee984a2e.cd3548","type":"trigger","z":"5d0da482.f44
17c","op1":"House Alarm Is
```

```
ON","op2":"","op1type":"str","op2type":"nul","duration":"60","extend":tru
e,"units":"s","reset":"","bytopic":"all","name":"","x":490,"y":560,"wires
":[["df04863e.7746a8","66e4c5c0.21ffac","99a8cbe9.315ae8"]]},{"id":"57749
220.1f54bc","type":"trigger","z":"5d0da482.f4417c","op1":"Cigar Humidor
Humidity
Low","op2":"","op1type":"str","op2type":"nul","duration":"60","extend":tr
ue,"units":"s","reset":"","bytopic":"all","name":"","x":470,"y":240,"wire
s":[["a3e3392f.9f10b8","1aa4e8a5.55f407"]]},{"id":"6c9e341.c5122cc","type
":"function","z":"5d0da482.f4417c","name":"Function
Compare","func":"context.slider = context.slider || 0.0;\ncontext.sensor
= context.sensor || 0.0;\n\nif (msg.topic === 'Humidity Slider') {\n
context.slider = msg.payload;\n} else if (msg.topic === '/sensor1/hum')
{\n  context.sensor = msg.payload;\n}\n\nif (context.sensor <
context.slider) {\n  return {topic: 'Humidor Alarm', payload: 'Low
Humidity'}\n} else {\n  return
null\n}","outputs":1,"noerr":0,"x":390,"y":400,"wires":[["57749220.1f54bc
","ad1ba6f6.acbaa8"]]},{"id":"a3e3392f.9f10b8","type":"blynk-ws-out-
notify","z":"5d0da482.f4417c","name":"Humidity
Notification","client":"55599e05.7bf55","queue":true,"rate":"10","x":740,
"y":160,"wires":[]},{"id":"1aa4e8a5.55f407","type":"blynk-api-out-
email","z":"5d0da482.f4417c","name":"Humidity Alarm E-
Mail","email":"u003rhg@gmail.com","client":"f9c90265.5b199","x":740,"y":2
40,"wires":[]},{"id":"97376f93.d151f","type":"amazon-echo-
hub","z":"5d0da482.f4417c","port":"8000","enableinput":false,"x":150,"y":
1980,"wires":[["d859e9ad.4cf5d8","e9a64f32.f3b81"]]},{"id":"d859e9ad.4cf5
d8","type":"amazon-echo-device","z":"5d0da482.f4417c","name":"Alarm
Siren","topic":"","x":450,"y":1900,"wires":[["369aa668.5154fa"]]},{"id":"
369aa668.5154fa","type":"change","z":"5d0da482.f4417c","name":"","rules":
[{"t":"change","p":"payload","pt":"msg","from":"on","fromt":"str","to":"0
","tot":"str"},{"t":"change","p":"payload","pt":"msg","from":"off","fromt
":"str","to":"1","tot":"str"}],"action":"","property":"","from":"","to":"
","reg":false,"x":700,"y":1880,"wires":[["348434fb.1a577c"]]},{"id":"f76e
d13f.ceb82","type":"change","z":"5d0da482.f4417c","name":"","rules":[{"t"
:"change","p":"payload","pt":"msg","from":"on","fromt":"str","to":"0","to
t":"str"},{"t":"change","p":"payload","pt":"msg","from":"off","fromt":"st
r","to":"1","tot":"str"}],"action":"","property":"","from":"","to":"","re
g":false,"x":380,"y":1780,"wires":[["25062ef6.6c2e02"]]},{"id":"e9a64f32.
f3b81","type":"amazon-echo-device","z":"5d0da482.f4417c","name":"Alarm
Horn","topic":"","x":230,"y":1860,"wires":[["f76ed13f.ceb82"]]},{"id":"6f
ba49cc.7e3018","type":"mqtt-broker","z":"","name":"PI
Broker","broker":"PI
Broker@192.168.1.31","port":"1883","clientid":"","usetls":false,"compatmo
de":true,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":
"0","birthRetain":"false","birthPayload":"","closeTopic":"","closeQos":"0
","closePayload":"","willTopic":"","willQos":"0","willPayload":""},{"id":
"f671b92d.bf2c58","type":"ui_group","z":"","name":"Temperature (Degrees
F)","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},{"id
":"9f88262d.a77a28","type":"ui_group","z":"","name":"% Relative
Humidity","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false
},{"id":"f9c90265.5b199","type":"blynk-api-client","z":"","name":"Blynk
Server","api":"http://blynk-cloud.com/","path":"ws://blynk-
cloud.com/websockets","key":"RAZA74k7icCDfFTu0D49XQ95mEoYf3a2"},{"id":"87
11b80a.b96c48","type":"ui_group","z":"","name":"Battery
Volts","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},{
```

```
"id":"f3d16070.eff6f","type":"ui_group","z":"","name":"AC
Power","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},{
"id":"f22674fd.edefc8","type":"ui_group","z":"","name":"Alarm","tab":"414
d8eb8.2e573","disp":true,"width":"6","collapse":false},{"id":"ae170b9.f25
eef8","type":"ui_group","z":"","name":"Armed","tab":"414d8eb8.2e573","dis
p":true,"width":"6","collapse":false},{"id":"2edc2351.f8cb4c","type":"ui_
group","z":"","name":"Alarm Horn
Switch","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},
{"id":"6689f280.8135ac","type":"ui_group","z":"","name":"Alarm Siren
Switch","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},
{"id":"a2fcc157.893b8","type":"ui_group","z":"","name":"Humidity
Slider","tab":"414d8eb8.2e573","disp":true,"width":"6","collapse":false},
{"id":"55599e05.7bf55","type":"blynk-ws-client","z":"","name":"Blynk
Websocket","path":"ws://blynk-
cloud.com/websockets","key":"RAZA74k7icCDfFTu0D49XQ95mEoYf3a2","dbg_all":
false,"dbg_read":false,"dbg_write":false,"dbg_notify":false,"dbg_mail":fa
lse,"dbg_prop":false,"dbg_sync":false,"dbg_bridge":false,"dbg_low":false,
"dbg_pins":"","multi_cmd":false,"proxy_type":"no","proxy_url":"","enabled
":true},{"id":"414d8eb8.2e573","type":"ui_tab","z":"","name":"Home","icon
":"dashboard","disabled":false,"hidden":false}]
```

```
"""
Wifi Manager Routine Revision 1 (Has an extra Field for a Broker Server IP address)
=================================================================================
"""

# Import Modules
# --------------
import network
import socket
import ure
import time
import os
import machine
from network import WLAN
from machine import Pin, I2C
from time import sleep
from ssd1306 import SSD1306_I2C

# Constants
# ---------
ap_ssid = "ESP832 Alarm WifiManager"
ap_password = "password"
ap_authmode = 3  # WPA2
WIFI_LED_PIN = 25 # ESP32 on board LED
OLED_RST_PIN = 16 # ESP32 OLED display Reset pin

# Initial Setup
# -------------
led = Pin(WIFI_LED_PIN, Pin.OUT, value=0) # Sets up LED and starts
OLED_RST = Pin(OLED_RST_PIN, Pin.OUT, value=1) # Sets up OLED RST to high
i2c = I2C(sda = Pin(4), scl = Pin(15)) # ESP32 OLED I2C Pins
display = SSD1306_I2C(128, 64, i2c) # ESP32 I2C display resolution

# Network Data File & Parameters
# ------------------------------
NETWORK_PROFILES = 'wifi.dat' # File to hold (SSID, PASS, Broker IP Address)
profiles = {} # Dictionary to hold keys & information
ssid1 = ""  # WiFi SSID
password = ""   # WiFi Password
broker_ip = ""  # Brooker IP Address

# Access Point and Staion Network Class Objects
# ---------------------------------------------
ap = WLAN(network.AP_IF)
sta = WLAN(network.STA_IF)

# Generic Function to Display 6 lines on OLED
# -------------------------------------
def Display_Stat(line1,line2,line3,line4,line5,line6,delaytime):
    display.fill(0) # Clear Previous Text
    display.text(line1,0,0,1) #Line 1
    display.text(line2,0,10,1) # Line 2
    display.text(line3,0,20,1) # Line 3
    display.text(line4,0,30,1) #Line 4
    display.text(line5,0,40,1) # Line 5
    display.text(line6,0,50,1) # Line 6
    display.show() # Display New Text
    sleep(delaytime) # Delay secs

# Function to Reboot the ESP32
# ----------------------------
def reboot():
    print("ESP32 is Rebooting....")
    L1 = "SSID and Pass"
    L2 = "Is Saved in a"
    L3 = ""
    L4 = "Configuration"
```

```
68          L5 = "File"
69          L6 = ""
70          delayx = 3
71          Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
72          machine.reset() # Hardware reset command
73
74      server_socket = None
75
76      # Function to return a working STA instance or "None" on the Main.py Program
77      # ---------------------------------------------------------------------------
78      def get_connection():
79          global ssid1, password, profiles
80          L1 = "Hello"
81          L2 = "Starting Wi-Fi"
82          L3 = "Let's Go..."
83          L4 = "Attemping to"
84          L5 = "Connect....."
85          L6 = "Please Wait"
86          delayx = 3
87          Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
88          if sta.isconnected(): # First check if there already is any connection:
89              led.value(1) # Turn on LED
90              return sta
91          connected = False # Means no connection previouly existed
92          try: # ESP32 WiFi takes time, wait a bit and try once more
93              time.sleep(5) # 5 second delay
94              if sta.isconnected(): # Check for a Wifi connection once more
95                  led.value(1) # Turn on LED
96                  return sta
97              # Read known network profiles from profiles file
98              profiles = read_profiles()
99              # Search WiFis networks in range
100             sta.active(True) # Activate station wifi
101             networks = sta.scan() # Scan all networks in area, and list them
102             AUTHMODE = {0: "open", 1: "WEP", 2: "WPA-PSK", 3: "WPA2-PSK", 4: "WPA/WPA2-PSK"}
103             for ssid, bssid, channel, rssi, authmode, hidden in sorted(networks, key=lambda
                x: x[3], reverse=True):
104                 ssid = ssid.decode('utf-8') # Convert encoding
105                 encrypted = authmode > 0 # See if wifi network is encrypted
106                 print("ssid: %s chan: %d rssi: %d authmode: %s" % (ssid, channel, rssi,
                    AUTHMODE.get(authmode, '?')))
107                 if encrypted:
108                     if ssid ==ssid1: # Check if scanned network is the same network in file
109                         ssid1 = profiles['SSID'] # SSID written from Web Page
110                         password = profiles['PASS'] # Password entered from web page
111                         print("ssid1 Read before connect = " + ssid1) # Debug
112                         print("password Read before connect = " + password) # Debug
113                         connected = do_connect(ssid1, password)
114                     else:
115                         print("skipping unknown encrypted network")
116                 else:  # open
117                     connected = do_connect(ssid1, None) # Unencrypted network
118                 if connected:
119                     break # exit if connected from file settings
120         except OSError as e:
121             print("exception", str(e))
122
123         # start web server for connection manager:
124         if not connected: # If not connected to wifi, start web server
125             connected = start() # Goto web server function
126         return sta if connected else None
127
128     # Function to read Network SSID Profiles
129     # --------------------------------------
130     def read_profiles():
131         global password, broker_ip, profiles, ssid1
132         with open(NETWORK_PROFILES) as f: # Open file and read settings
```

```
133             lines = f.readlines()
134         profiles = {}
135         for line in lines:
136             ssid1, password, broker_ip = line.strip("\n").split(";") # Look for this
                information
137             profiles['SSID'] = ssid1 # Web Page SSID stored
138             profiles['PASS'] = password # Web Page Password stored
139             profiles['IP'] = broker_ip # Web Page Broker server IP address stored
140         print("File Read after strip = " + str(lines)) # Debug
141         print("File Read Profiles = " + str(profiles)) # Debug
142         return profiles
143 
144     # Function to Write Network SSID Profiles
145     # ---------------------------------------
146     def write_profiles(profiles): # Write values entered from web page
147         global ssid1, password, broker_ip
148         lines = []
149         lines.append("%s;%s;%s\n" % (ssid1, password, broker_ip)) # Join information together
150         with open(NETWORK_PROFILES, "w") as f:
151             f.write(''.join(lines))
152         print("File Write = " + str(lines)) # Debug
153 
154     # Function to connect to WiFi
155     # ---------------------------
156     def do_connect(ssid1, password):
157         sta.active(True)
158         if sta.isconnected():  # If connected return
159             return None
160         print('Trying to connect to %s...' % ssid1)
161         sta.connect(ssid1, password)
162         attempt = 0
163         for retry in range(12): # Try to connect 12 up to times at 1 second intervals
164             connected = sta.isconnected()
165             if connected: # Leave function when connection is established
166                 break
167             L1 = "Trying to"
168             L2 = "Connect....."
169             L3 = ""
170             L4 = "Attempt: " + str(attempt)
171             L5 = ""
172             L6 = ""
173             delayx = 1
174             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
175             attempt += 1
176             print("Attempts = " + str(attempt))
177         if connected:
178             print('\nConnected. Network config: ', sta.ifconfig())
179             led.value(1) # Turn on LED
180             #ap.active(False) # Turn off AP Mode
181             L1 = "WiFi Connected!"
182             L2 = "IP Address Is:"
183             L3 = str(sta.ifconfig())
184             L4 = ""
185             L5 = "Returning to"
186             L6 = "Main Program"
187             delayx = 3
188             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
189         else:
190             print('\nFailed. Not Connected to: ' + ssid1)
191             led.value(0) # Turn off LED
192             L1 = "Wi-Fi"
193             L2 = "Connection"
194             L3 = "Failed"
195             L4 = ""
196             L5 = ""
197             L6 = ""
198             delayx = 3
```

```
            Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
        return connected

    # Function to Send Web Page Total Response
    # ----------------------------------------
    def send_response(client, payload, status_code=200):
        client.sendall("HTTP/1.0 {} OK\r\n".format(status_code))
        client.sendall("Content-Type: text/html\r\n")
        client.sendall("Content-Length: {}\r\n".format(len(payload)))
        client.sendall("\r\n")
        if len(payload) > 0:
            client.sendall(payload)

    # Function to Display HTML Web Browser
    # ------------------------------------
    def handle_root(client): # Create web Page with fields to enter data
        sta.active(True)
        ssids = sorted(ssid.decode('utf-8') for ssid, *_ in sta.scan())
        response = []
        response.append("""\
            <html>
              <body bgcolor="#00FFFF">
                <h1 style="color: #5e9ca0; text-align: center;">
                    <span style="color: #ff0000;">
                        ESP32 WiFi Setup
                    </span>
                </h1>
                <form action="configure" method="post">
                    <table style="margin-left: auto; margin-right: auto;">
                        <tbody>
        """)
        for ssid in ssids:
            response.append("""\
                            <tr>
                                <td colspan="2">
                                    <input type="radio" name="ssid" value="{0}" />{0}
                                </td>
                            </tr>
            """.format(ssid))

        response.append("""\
                            <tr>
                                <td>Password:</td>
                                <td><input name="password" type="password" /></td>
                            </tr>
                            <tr>
                                <td>Brooker IP Address:</td>
                                <td><input name="broker_ip" type="text" /></td>
                            </tr>
                        </tbody>
                    </table>
                    <p style="text-align: center;">
                        <input type="submit" value="Submit" />
                    </p>
                </form>
                <p> </p>
                <hr />
                <h5>
                    <span style="color: #ff0000;">
                        Your ssid and password information will be saved into the
                        "%(filename)s" file in your ESP module for future usage.
                        Be careful about security!
                    </span>
                </h5>
                <hr />
              </body>
            </html>
```

```
        """ % dict(filename=NETWORK_PROFILES))
        send_response(client, "\n".join(response))

    # Function to Get SSID & Password & Broker Server IP Address:
    # ------------------------------------------------------------
    def handle_configure(client, request):
        global ssid1, password, broker_ip
        # Search returned "POST" information from HTML server request
        match = ure.search("ssid=([^&]*)&password=([^&].*)&broker_ip=([^&].*)", request)
        if match is None: # Go back if no data (return)
            send_response(client, "Parameters not found", status_code=400)
            print("Bad handle_configure function") # Debug
            return False
        try: # Store "POST" information from HTML server request into a global variable
            ssid1 = match.group(1).decode("utf-8").replace("%3F", "?").replace("%21", "!")
            password = match.group(2).decode("utf-8").replace("%3F", "?").replace("%21", "!")
            broker_ip = match.group(3).decode("utf-8") # Roy added brooker_ip and (3)
        except:
            ssid1 = match.group(1).replace("%3F", "?").replace("%21", "!")
            password = match.group(2).replace("%3F", "?").replace("%21", "!")
            broker_ip = match.group(3)
        if len(ssid1) == 0: # Check if no data
            send_response(client, "SSID must be provided", status_code=400)
            return False
        if do_connect(ssid1, password): # Try to connect to network with global variables
            response = """\
                <html>
                    <center>
                        <br><br>
                        <h1 style="color: #5e9ca0; text-align: center;">
                            <span style="color: #ff0000;">
                                ESP32 successfully connected to WiFi network %(ssid)s.
                            </span>
                        </h1>
                        <br><br>
                    </center>
                </html>
            """ % dict(ssid=ssid1)
            send_response(client, response)
            try:
                profiles = read_profiles() # Read data from function if it exists
            except OSError: # Write data to files if it does not exist, so it will read
            data on power up
                profiles = {}
            profiles['SSID'] = ssid1 # Store SSID to dictionary key
            profiles['PASS'] = password # Store Password to dictionary key
            profiles['IP'] = broker_ip # Store Broker Server IP Address to dictionary key
            write_profiles(profiles) # Write data to file function
            return True
        else:
            response = """\
                <html>
                    <center>
                        <h1 style="color: #5e9ca0; text-align: center;">
                            <span style="color: #ff0000;">
                                ESP32 could not connect to WiFi network %(ssid)s.
                            </span>
                        </h1>
                        <br><br>
                        <form>
                            <input type="button" value="Go back!"
                            onclick="history.back()"></input>
                        </form>
                    </center>
                </html>
            """ % dict(ssid=ssid1)
            send_response(client, response)
```

```
331             return False
332
333     # Path Error Function
334     # -------------------
335     def handle_not_found(client, url):
336         send_response(client, "Path not found: {}".format(url), status_code=404)
337
338     # Web Socket Close Function
339     # -------------------------
340     def stop():
341         global server_socket
342         if server_socket:
343             server_socket.close()
344             server_socket = None
345
346     # Web Server Start Function
347     # -------------------------
348     def start(port=80): # Start Web Socket and ESP32 communication using an Access Point to
        connect a device
349         global server_socket
350         addr = socket.getaddrinfo('0.0.0.0', port)[0][-1]
351         stop() # Goto Function
352         sta.active(True)
353         ap.active(True)
354         ap.config(essid=ap_ssid, password=ap_password, authmode=ap_authmode)
355         server_socket = socket.socket()
356         server_socket.bind(addr)
357         server_socket.listen(1)
358         print('Connect to WiFi ssid ' + ap_ssid + ', default password: ' + ap_password)
359         print('and access the ESP via your favorite web browser at 192.168.4.1.')
360         print('Listening on:', addr)
361         L1 = "Server Started"
362         L2 = "Connect to AP"
363         L3 = "Type 'password'"
364         L4 = "Open Web Browser"
365         L5 = "Type in Address"
366         L6 = "192.168.4.1"
367         delayx = 3
368         Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
369         while True:
370             if sta.isconnected():
371                 return True
372             client, addr = server_socket.accept()
373             print('client connected from', addr)
374             try:
375                 client.settimeout(5.0)
376                 request = b""
377                 try:
378                     while "\r\n\r\n" not in request:
379                         request += client.recv(512)
380                 except OSError:
381                     pass
382                 print("Request is: {}".format(request))
383                 if "HTTP" not in request:
384                     # skip invalid requests
385                     continue
386                 # version 1.9 compatibility
387                 try:
388                     url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                        request).group(1).decode("utf-8").rstrip("/")
389                 except:
390                     url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                        request).group(1).rstrip("/")
391                 print("URL is {}".format(url))
392                 if url == "":
393                     handle_root(client)
394                 elif url == "configure":
```

```
                handle_configure(client, request)
            else:
                handle_not_found(client, url)
        finally:
            client.close()
            #reboot() # Goto Reboot function (Not Utilized)


```

```
"""
Wifi Manager Routine Revision 2 (Has an extra Field for a Broker Server IP address)
This version is for ESP8266 with a 128X32 OLED Display
================================================================================
"""

# Import Modules
# --------------
import network
import socket
import ure
import time
import os
import machine
from network import WLAN
from machine import Pin, I2C
from time import sleep
from ssd1306 import SSD1306_I2C

# Constants
# ---------
ap_ssid = "ESP8266 Humidor WifiManager"
ap_password = "password"
ap_authmode = 3  # WPA2
WIFI_LED_PIN = 2 # ESP8266 on board LED
#OLED_RST_PIN = 16 # ESP8266 OLED display Reset pin

# Initial Setup
# -------------
led = Pin(WIFI_LED_PIN, Pin.OUT, value=1) # Sets up LED and starts off
#OLED_RST = Pin(OLED_RST_PIN, Pin.OUT, value=1) # Sets up OLED RST to high
i2c = I2C(-1, sda = Pin(4), scl = Pin(5)) # Set Up OLED I2C Pins with software mode
display = SSD1306_I2C(128, 32, i2c) # I2C parameters for OLED Display

# Network Data File & Parameters
# ------------------------------
NETWORK_PROFILES = 'wifi.dat' # File to hold (SSID, PASS, Broker IP Address)
profiles = {} # Dictionary to hold keys & information
ssid1 = ""  # WiFi SSID
password = ""   # WiFi Password
broker_ip = ""  # Brooker IP Address

# Access Point and Staion Network Class Objects
# ---------------------------------------------
ap = WLAN(network.AP_IF)
sta = WLAN(network.STA_IF)

# Generic Function to Display 6 lines on OLED
# -------------------------------------------
def Display_Stat(line1,line2,line3,delaytime):
    display.fill(0) # Clear Previous Text
    display.text(line1,0,0,1) #Line 1
    display.text(line2,0,10,1) # Line 2
    display.text(line3,0,20,1) # Line 3
    display.show() # Display New Text
    sleep(delaytime) # Delay secs

# Function to Reboot the ESP8266
# ------------------------------
def reboot():
    print("ESP8266 Is Rebooting.....")
    L1 = "ESP8266 Is"
    L2 = "Rebooting......."
    L3 = ""
    delayx = 3
    Display_Stat(L1,L2,L3,delayx) # Goto Display function
    machine.reset() # Hardware reset command
```

```
68
69  server_socket = None
70
71  # Function to return a working STA instance or "None" on the Main.py Program
72  # ---------------------------------------------------------------------------
73  def get_connection():
74      global ssid1, password, profiles
75      L1 = "Hello"
76      L2 = "Starting Wi-Fi"
77      L3 = "Connecting......"
78      delayx = 3
79      Display_Stat(L1,L2,L3,delayx) # Goto Display function
80      if sta.isconnected(): # First check if there already is any connection:
81          led.value(0) # Turn on LED
82          return sta
83      connected = False # Means no connection previouly existed
84      try: # ESP8266 WiFi takes time, wait a bit and try once more
85          time.sleep(5) # 5 second delay
86          if sta.isconnected(): # Check for a Wifi connection once more
87              led.value(0) # Turn on LED
88              return sta
89          # Read known network profiles from profiles file
90          profiles = read_profiles()
91          # Search WiFis networks in range
92          sta.active(True) # Activate station wifi
93          networks = sta.scan() # Scan all networks in area, and list them
94          AUTHMODE = {0: "open", 1: "WEP", 2: "WPA-PSK", 3: "WPA2-PSK", 4: "WPA/WPA2-PSK"}
95          for ssid, bssid, channel, rssi, authmode, hidden in sorted(networks, key=lambda
            x: x[3], reverse=True):
96              ssid = ssid.decode('utf-8') # Convert encoding
97              encrypted = authmode > 0 # See if wifi network is encrypted
98              print("ssid: %s chan: %d rssi: %d authmode: %s" % (ssid, channel, rssi,
                AUTHMODE.get(authmode, '?')))
99              if encrypted:
100                 if ssid ==ssid1: # Check if scanned network is the same network in file
101                     ssid1 = profiles['SSID'] # SSID written from Web Page
102                     password = profiles['PASS'] # Password entered from web page
103                     print("ssid1 Read before connect = " + ssid1) # Debug
104                     print("password Read before connect = " + password) # Debug
105                     connected = do_connect(ssid1, password)
106                 else:
107                     print("skipping unknown encrypted network")
108             else:  # open
109                 connected = do_connect(ssid1, None) # Unencrypted network
110             if connected:
111                 break # exit if connected from file settings
112     except OSError as e:
113         print("exception", str(e))
114
115     # start web server for connection manager:
116     if not connected: # If not connected to wifi, start web server
117         connected = start() # Goto web server function
118     return sta if connected else None
119
120 # Function to read Network SSID Profiles
121 # --------------------------------------
122 def read_profiles():
123     global password, broker_ip, profiles, ssid1
124     with open(NETWORK_PROFILES) as f: # Open file and read settings
125         lines = f.readlines()
126     profiles = {}
127     for line in lines:
128         ssid1, password, broker_ip = line.strip("\n").split(";") # Look for this
            information
129         profiles['SSID'] = ssid1 # Web Page SSID stored
130         profiles['PASS'] = password # Web Page Password stored
131         profiles['IP'] = broker_ip # Web Page Broker server IP address stored
```

```
        print("File Read after strip = " + str(lines)) # Debug
        print("File Read Profiles = " + str(profiles)) # Debug
        return profiles

    # Function to Write Network SSID Profiles
    # ---------------------------------------
    def write_profiles(profiles): # Write values entered from web page
        global ssid1, password, broker_ip
        lines = []
        lines.append("%s;%s;%s\n" % (ssid1, password, broker_ip)) # Join information together
        with open(NETWORK_PROFILES, "w") as f:
            f.write(''.join(lines))
        print("File Write = " + str(lines)) # Debug

    # Function to connect to WiFi
    # ---------------------------
    def do_connect(ssid1, password):
        sta.active(True)
        if sta.isconnected():  # If connected return
            return None
        print('Trying to connect to %s...' % ssid1)
        sta.connect(ssid1, password)
        attempt = 0
        for retry in range(16): # Try to connect up to 16 times at 1 second intervals
            connected = sta.isconnected()
            if connected: # Leave function when connection is established
                break
            L1 = "Trying to"
            L2 = "Connect....."
            L3 = "Attempt: " + str(attempt)
            delayx = 1
            Display_Stat(L1,L2,L3,delayx) # Goto Display function
            attempt += 1
            print("Attempts = " + str(attempt))
        if connected:
            print('\nConnected. Network config: ', sta.ifconfig())
            led.value(0) # Turn on LED
            #ap.active(False) # Turn off AP Mode
            L1 = "WiFi Connected!"
            L2 = "IP Address Is:"
            L3 = str(sta.ifconfig())
            delayx = 2
            Display_Stat(L1,L2,L3,delayx) # Goto Display function
        else:
            print('\nFailed. Not Connected to: ' + ssid1)
            led.value(1) # Turn off LED
            L1 = "Wi-Fi"
            L2 = "Connection"
            L3 = "Failed"
            delayx = 3
            Display_Stat(L1,L2,L3,delayx) # Goto Display function
        return connected

    # Function to Send Web Page Total Response
    # ----------------------------------------
    def send_response(client, payload, status_code=200):
        client.sendall("HTTP/1.0 {} OK\r\n".format(status_code))
        client.sendall("Content-Type: text/html\r\n")
        client.sendall("Content-Length: {}\r\n".format(len(payload)))
        client.sendall("\r\n")
        if len(payload) > 0:
            client.sendall(payload)

    # Function to Display HTML Web Browser
    # ------------------------------------
    def handle_root(client): # Create web Page with fields to enter data
        sta.active(True)
```

```
    ssids = sorted(ssid.decode('utf-8') for ssid, *_ in sta.scan())
    response = []
    response.append("""\
        <html>
          <body bgcolor="#00FFFF">
            <h1 style="color: #5e9ca0; text-align: center;">
                <span style="color: #ff0000;">
                    ESP8266 WiFi Setup
                </span>
            </h1>
            <form action="configure" method="post">
                <table style="margin-left: auto; margin-right: auto;">
                    <tbody>
    """)
    for ssid in ssids:
        response.append("""\
                        <tr>
                            <td colspan="2">
                                <input type="radio" name="ssid" value="{0}" />{0}
                            </td>
                        </tr>
        """.format(ssid))

    response.append("""\
                        <tr>
                            <td>Password:</td>
                            <td><input name="password" type="password" /></td>
                        </tr>
                        <tr>
                            <td>Brooker IP Address:</td>
                            <td><input name="broker_ip" type="text" /></td>
                        </tr>
                    </tbody>
                </table>
                <p style="text-align: center;">
                    <input type="submit" value="Submit" />
                </p>
            </form>
            <p> </p>
            <hr />
            <h5>
                <span style="color: #ff0000;">
                    Your ssid and password information will be saved into the
                    "%(filename)s" file in your ESP module for future usage.
                    Be careful about security!
                </span>
            </h5>
            <hr />
          </body>
        </html>
    """ % dict(filename=NETWORK_PROFILES))
    send_response(client, "\n".join(response))

# Function to Get SSID & Password & Broker Server IP Address:
# -----------------------------------------------------------
def handle_configure(client, request):
    global ssid1, password, broker_ip
    # Search returned "POST" information from HTML server request
    match = ure.search("ssid=([^&]*)&password=([^&].*)&broker_ip=([^&].*)", request)
    if match is None: # Go back if no data (return)
        send_response(client, "Parameters not found", status_code=400)
        print("Bad handle_configure function") # Debug
        return False
    try: # Store "POST" information from HTML server request into a global variable
        ssid1 = match.group(1).decode("utf-8").replace("%3F", "?").replace("%21", "!")
        password = match.group(2).decode("utf-8").replace("%3F", "?").replace("%21", "!")
        broker_ip = match.group(3).decode("utf-8") # Roy added brooker_ip and (3)
```

```
266         except:
267             ssid1 = match.group(1).replace("%3F", "?").replace("%21", "!")
268             password = match.group(2).replace("%3F", "?").replace("%21", "!")
269             broker_ip = match.group(3)
270         if len(ssid1) == 0: # Check if no data
271             send_response(client, "SSID must be provided", status_code=400)
272             return False
273         if do_connect(ssid1, password): # Try to connect to network with global variables
274             response = """\
275                 <html>
276                     <center>
277                         <br><br>
278                         <h1 style="color: #5e9ca0; text-align: center;">
279                             <span style="color: #ff0000;">
280                                 ESP8266 successfully connected to WiFi network %(ssid)s.
281                             </span>
282                         </h1>
283                         <br><br>
284                     </center>
285                 </html>
286             """ % dict(ssid=ssid1)
287             send_response(client, response)
288             try:
289                 profiles = read_profiles() # Read data from function if it exists
290             except OSError: # Write data to files if it does not exist, so it will read
    data on power up
291                 profiles = {}
292             profiles['SSID'] = ssid1 # Store SSID to dictionary key
293             profiles['PASS'] = password # Store Password to dictionary key
294             profiles['IP'] = broker_ip # Store Broker Server IP Address to dictionary key
295             write_profiles(profiles) # Write data to file function
296             return True
297         else:
298             response = """\
299                 <html>
300                     <center>
301                         <h1 style="color: #5e9ca0; text-align: center;">
302                             <span style="color: #ff0000;">
303                                 ESP8266 could not connect to WiFi network %(ssid)s.
304                             </span>
305                         </h1>
306                         <br><br>
307                         <form>
308                             <input type="button" value="Go back!"
    onclick="history.back()"></input>
309                         </form>
310                     </center>
311                 </html>
312             """ % dict(ssid=ssid1)
313             send_response(client, response)
314             return False
315 
316     # Path Error Function
317     # -------------------
318     def handle_not_found(client, url):
319         send_response(client, "Path not found: {}".format(url), status_code=404)
320 
321     # Web Socket Close Function
322     # -------------------------
323     def stop():
324         global server_socket
325         if server_socket:
326             server_socket.close()
327             server_socket = None
328 
329     # Web Server Start Function
330     # -------------------------
```

```
def start(port=80): # Start Web Socket and ESP8266 communication using an Access Point to connect a device
    global server_socket
    addr = socket.getaddrinfo('0.0.0.0', port)[0][-1]
    stop() # Goto Function
    sta.active(True)
    ap.active(True)
    ap.config(essid=ap_ssid, password=ap_password, authmode=ap_authmode)
    server_socket = socket.socket()
    server_socket.bind(addr)
    server_socket.listen(1)
    print('Connect to WiFi ssid ' + ap_ssid + ', default password: ' + ap_password)
    print('and access the ESP via your favorite web browser at 192.168.4.1.')
    print('Listening on:', addr)
    L1 = "Server Started"
    L2 = "Connect to AP"
    L3 = "Type 'password'"
    delayx = 3
    Display_Stat(L1,L2,L3,delayx) # Goto Display function
    L4 = "Open Web Browser"
    L5 = "Type in Address"
    L6 = "192.168.4.1"
    delayx = 3
    Display_Stat(L4,L5,L6,delayx) # Goto Display function
    while True:
        if sta.isconnected():
            return True
        client, addr = server_socket.accept()
        print('client connected from', addr)
        try:
            client.settimeout(5.0)
            request = b""
            try:
                while "\r\n\r\n" not in request:
                    request += client.recv(512)
            except OSError:
                pass
            print("Request is: {}".format(request))
            if "HTTP" not in request:
                # skip invalid requests
                continue
            # version 1.9 compatibility
            try:
                url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                request).group(1).decode("utf-8").rstrip("/")
            except:
                url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                request).group(1).rstrip("/")
            print("URL is {}".format(url))
            if url == "":
                handle_root(client)
            elif url == "configure":
                handle_configure(client, request)
            else:
                handle_not_found(client, url)
        finally:
            client.close()
            #reboot() # Goto Reboot function (Not Utilized)


```