from inicequip import \*

import paho.mqtt.client as paho

import threading

import time

client = paho.Client()

client.connect('iot.eclipse.org', 1883)

client.loop\_start()

class ThreadPer(threading.Thread):

def \_\_init\_\_(self, RPM, minTempo, Sentido, Revolucao, Repetir, ser1):

threading.Thread.\_\_init\_\_(self)

self.iterations = 0

self.daemon = True # OK for main to exit even if instance is still running

self.paused = True # start out paused

self.stopped = False

self.state = threading.Condition()

self.RPM = RPM

self.minTempo = minTempo

self.Sentido = Sentido

self.Revolucao = Revolucao

self.Repetir = Repetir

self.ser1 = ser1

self.first = True

def run(self):

self.resume() # unpause self

while True:

with self.state:

if self.paused:

self.state.wait() # block until notified

if self.stopped:

break

# do stuff

if self.iterations < len(self.RPM):

if self.RPM[self.iterations] >= 10:

texto = "\x02P01S{0}{1}V{2}G\x0d".format(self.Sentido[self.iterations], self.RPM[self.iterations], round(self.Revolucao[self.iterations],2))

self.ser1.write(texto.encode("ascii"))

else:

self.ser1.write("\x02P01H\x0d".encode("ascii"))

time.sleep(self.minTempo[self.iterations]\*60)

self.iterations += 1

else:

if self.Repetir == True:

self.iterations = 0

else:

self.stop()

def resume(self):

if self.first == False:

texto = "\x02P01S{0}{1}V{2}G\x0d".format(self.Sentido[self.iterations], self.RPM[self.iterations], round(self.Revolucao[self.iterations],2))

self.ser1.write(texto.encode("ascii"))

self.first = False

with self.state:

self.paused = False

self.state.notify() # unblock self if waiting

def pause(self):

self.ser1.write("\x02P01H\x0d".encode("ascii"))

with self.state:

self.paused = True # make self block and wait

def stop(self):

self.ser1.write("\x02P01H\x0d".encode("ascii"))

self.ser1.close()

with self.state:

self.stopped = True

class ThreadInc(threading.Thread):

def \_\_init\_\_(self, Temperatura, minTempo2, Repetir2, ser2):

threading.Thread.\_\_init\_\_(self)

self.iterations = 0

self.daemon = True # OK for main to exit even if instance is still running

self.paused = True # start out paused

self.stopped = False

self.state = threading.Condition()

self.Temperatura = Temperatura

self.minTempo2 = minTempo2

self.Repetir2 = Repetir2

self.ser2 = ser2

self.first = True

def run(self):

self.resume() # unpause self

while True:

with self.state:

if self.paused:

self.state.wait() # block until notified

if self.stopped:

break

# do stuff

if self.iterations < len(self.Temperatura):

texto2 = "PT{0}\x0d".format(round(self.Temperatura[self.iterations],1))

self.ser2.write(texto2.encode("ascii"))

time.sleep(self.minTempo2[self.iterations]\*60)

self.iterations += 1

else:

if self.Repetir2 == True:

self.iterations = 0

else:

self.stop()

def resume(self):

if self.first == False:

texto2 = "PT{0}\x0d".format(round(self.Temperatura[self.iterations],1))

self.ser2.write(texto2.encode("ascii"))

self.first = False

with self.state:

self.paused = False

self.state.notify() # unblock self if waiting

def pause(self):

self.ser2.write("PT0\x0d".encode("ascii"))

with self.state:

self.paused = True # make self block and wait

def stop(self):

self.ser2.write("PT0\x0d".encode("ascii"))

self.ser2.close()

with self.state:

self.stopped = True

def listabomba(porta, RPM, Tempo, Sentido, Unidade, Repetir):

minTempo = []

ser1 = masterflex775030(porta)

ser1.write("\x02P01Z\x0d".encode("ascii"))

Revolucao = [0]\*len(RPM)

for i in range(len(RPM)):

if Unidade[i] == "seconds":

minTempo.append(Tempo[i]/60)

elif Unidade[i] == "hours":

minTempo.append(Tempo[i]\*60)

else:

minTempo.append(Tempo[i])

Revolucao[i] = RPM[i]\*minTempo[i]

funportempo = ThreadPer(RPM, minTempo, Sentido, Revolucao, Repetir, ser1)

funportempo.start()

mqtt = iotbp(ser1)

mqtt.start()

return funportempo, mqtt

def listaincubadora(porta2, Temperatura, Tempo2, Unidade2, Repetir2):

minTempo2 = []

ser2 = galaxy170r(porta2)

x2 = 0

for i in range(len(Tempo2)):

if Unidade2[i] == "hours":

minTempo2.append(Tempo2[i]\*60)

else:

minTempo2.append(Tempo2[i])

funportempo2 = ThreadInc(Temperatura, minTempo2, Repetir2, ser2)

funportempo2.start()

mqtt2 = iotin(ser2)

mqtt2.start()

return funportempo2, mqtt2

class iotbp(threading.Thread):# Testar

def \_\_init\_\_(self, ser1):

threading.Thread.\_\_init\_\_(self)

time.sleep(2)

self.ser1 = ser1

self.state = threading.Condition()

self.iterations = 0

self.daemon = True

self.paused = True

self.nome = ''

self.arq = ''

self.inter = 0

def run(self):

while self.ser1.is\_open == True:

if self.ser1.out\_waiting == False:

self.ser1.flushInput()

self.ser1.write("\x02P01S\x0d".encode("ascii"))

time.sleep(1)

resposta = self.ser1.read(self.ser1.inWaiting())

resposta = resposta.decode()

if len(resposta) > 9:

resposta = resposta[1:10]

bprpm = float(resposta[3:8])

if resposta[2:3] == "+":

bpdir = "Clockwise"

elif resposta[2:3] == "-":

bpdir = "Anticlockwise"

else:

bpdir = " "

(rc, mid) = client.publish("EvaCairoBPRPM", bprpm, qos=1)

(rc1,mid1) = client.publish("EvaCairoBPDir", bpdir, qos=1)

with self.state:

if self.paused:

pass

else:

if self.iterations == (self.inter - 1):#colcar um valor recebido do gui aqui

self.arq.write(time.strftime("%Y-%m-%d", time.localtime()) + "\t" + time.strftime("%X", time.localtime()) + "\t" + bpdir + "\t" + str(bprpm) + "\n")

self.iterations = 0

else:

self.iterations += 1

else:

(rc, mid) = client.publish("EvaCairoBPRPM", 0, qos=1)

def resume(self, inter):

self.nome = "Peristaltic Pump " + time.strftime("%Y-%m-%d", time.localtime())

self.arq = open("Data\\" + self.nome + ".tsv", 'w')

self.arq.write("Data\tTime\tDirection\tRPM\n")

self.inter = inter

with self.state:

self.paused = False

self.state.notify() # unblock self if waiting

def pause(self):

self.arq.close()

with self.state:

self.paused = True # make self block and wait

class iotinc(threading.Thread):

def \_\_init\_\_(self, ser2):

threading.Thread.\_\_init\_\_(self)

time.sleep(2)

self.ser2 = ser2

self.state = threading.Condition()

self.iterations = 0

self.daemon = True

self.paused = True

self.nome = ''

self.arq = ''

self.inter = 0

def run(self):

while self.ser2.is\_open == True:

if self.ser2.out\_waiting == False:

self.ser2.flushInput()

self.ser2.write("S\x0d".encode("ascii"))

self.ser2.readline()

self.ser2.readline()

resposta = self.ser2.readline()

inctemp = resposta[20:24].decode()

(rc, mid) = client.publish("EvaCairoInTemp", inctemp, qos=1)

with self.state:

if self.paused:

pass

else:

if self.iterations == 10:#colcar um valor recebido do gui aqui

self.arq.write(time.strftime("%Y-%m-%d", time.localtime()) + "\t" + time.strftime("%X", time.localtime()) + "\t" + inctemp + "\n")

self.iterations = 0

else:

self.iterations += 1

else:

(rc, mid) = client.publish("EvaCairoInTemp", 0, qos=1)

def resume(self):

self.nome = "Incubator " + time.strftime("%Y-%m-%d", time.localtime())

self.arq = open("Data\\" + self.nome + ".tsv", 'w')

self.arq.write("Data\tTime\tºC\n")

with self.state:

self.paused = False

self.state.notify() # unblock self if waiting

def pause(self):

self.arq.clode()

with self.state:

self.paused = True # make self block and wait