|  |  |  |  |
| --- | --- | --- | --- |
| **Aktion** | **Klasse** | **Methode** | **Übergabeparameter** |
| Erstellen eines Temperatursensorobjekts | Ds18b20 | \_\_init\_\_(w1\_address) | Seriennummer des Sensors |
| Erstellen eines Cloudspeicherobjektes | Cloud |  | URL des MQTT-Brokers, Basistopic und Quelle für Raumtemperatur |
| Setzen der Solltemperatur im Cloudobjekt |  | set\_setpointTemp(setpointTemp) |  |
| Setzen der Temperaturdifferenz für Hysterese im Cloudobjekt |  |  |  |
| Setzen der GPIO Nummer für Lüfterelektronik im Cloudobjekt |  |  |  |
| GPIO-Port im Cloudspeicher abfragen |  |  | --- |
| Lüfterobjekt erzeugen | Fan | \_\_init\_\_(gpio) | GPIO-Port |
| Reglerobjekt erzeugen |  |  | Cloudspeicher- und Lüfterobjekt |
| Aktuelle Raumtemperatur messen,  in der Cloud speichern | Cloud | update() | --- |
| Regeln |  |  | --- |

# Bisheriger Stand zur Kodierung der Klassen

import RPi.GPIO as GPIO

import paho.mqtt.publish as publish

import paho.mqtt.subscribe as subscribe

import time

class Controller():

    def \_\_init\_\_(self, cloud, fan):

        self.\_\_cloud=cloud # Konfigurationen und Werte

        self.\_\_fan=fan # Der Aktor für den Zweipunktregler

        self.\_\_splow=self.\_\_cloud.get\_setpointTemp() # untere Grenze

        self.\_\_hyst=self.\_\_cloud.get\_hyst() #Hysterese speichern

        self.\_\_sphigh=self.\_\_splow+self.\_\_hyst # obere Grenze

    def control(self):

        # Vergleich Ist- und Solltemperatur und Kühlung ansteuern

        # hier ist der Regler als Zweipunktregler mit Hysterese implementiert

        temperature = self.\_\_cloud.get\_roomTemp() # Aktuelle Temperatur aus der Cloud lesen

        if temperature <= self.\_\_splow:

            self.\_\_fan.off() # Kühlung ausschalten, wenn es kalt ist

        elif temperature > self.\_\_sphigh:

            self.\_\_fan.on() # Kühlung einschalten, wenn es zu warm wird

        return

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ende class Regelstrecke

class Fan():

    def \_\_init\_\_(self, gpio):

        self.\_\_gpio = gpio # GPIO Nr. des Lüfters

        GPIO.setmode(GPIO.BCM) # mit GPIO Bezeichnungen arbeiten

        GPIO.setup(self.\_\_gpio, GPIO.OUT) # Fan GPIO als Ausgang schalten

        self.off() # Lüfter per default ausschalten

        self.\_\_state = False # Zustand merken für Debuggingzwecke (siehe main())

    def on(self):

        GPIO.output(self.\_\_gpio, 1) # GPIO.HIGH, Lüfter an

        self.\_\_state = True # Zustand merken

        return

    def off(self):

        GPIO.output(self.\_\_gpio, 0) # GPIO.LOW, Lüfter aus

        self.\_\_state = False # Zustand merken

        return

    def get\_state(self):

        return self.\_\_state

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ende class Fan

class Ds18b20():

    def \_\_init\_\_(self, w1\_address):

        self.\_\_w1\_address=w1\_address

        self.\_\_w1\_dir = '/sys/bus/w1/devices/' + self.\_\_w1\_address + '/w1\_slave'

        self.\_\_tempCelsius = 255.0

    def get\_celsius(self) :

        # cat self.w1\_dir würde vom DS18B20 liefern:

        # 75 01 4b 46 7f ff 0b 10 78 : crc=78 YES

        # 75 01 4b 46 7f ff 0b 10 78 t=23312

        # Die Zahl hinter t= entspricht der Temperatur in tausendstel °C

        f = open(self.\_\_w1\_dir, 'r') # "Datei" öffnen

        lines = f.readlines()

        # Warten bis in der ersten Zeile ein YES erscheint. Dann sind die Daten gültig

        while lines[0].strip()[-3:] != 'YES':

            time.sleep(0.2)

            lines = f.readlines()

        tempStr = lines[1].find('t=')

        if tempStr != -1:

            # einen richtigen Wert gefunden

            self.\_\_tempCelsius = float(lines[1][tempStr+2:]) / 1000.0

        else:

            self.\_\_tempCelsius = 255.0

        return self.\_\_tempCelsius

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ende class Ds18b20

class Cloud():

    # in dieser Klasse realisiert mit einem MQTT Broker

    def \_\_init\_\_(self, hostname, basetopic, tempsen):

        self.\_\_roomTemp=255.0 # Isttemperatur

        self.\_\_setpointTemp=255.0 # Solltemperatur

        self.\_\_hyst=255.0 #Hysterese

        self.\_\_gpioFan=0 # GPIO BCM-Nr zur Lüfteransteuerung

        self.\_\_hostname=hostname # Name/ IP des MQTT Brokers

        basetopic = basetopic + "/" # Basistopic für das Abspeichern der Werte

        self.\_\_topicRoomTemp = basetopic + "roomtemp" # Topic für die Raumtemperatur

        self.\_\_topicSetpointTemp = basetopic + "setpointtemp" # Topic für die Solltemperatur

        self.\_\_topicHyst = basetopic + "hyst" # Topic für die Hysterese

        self.\_\_topicGpioFan = basetopic + "gpiofan" # Topic für die GPIO Nr. des Lüfters

        self.\_\_tempsen=tempsen # Der benutzte Temperatursensor

    def get\_roomTemp(self): # Serverraumtemperatur aus der Cloud lesen

        self.\_\_roomTemp = float((subscribe.simple(self.\_\_topicRoomTemp, hostname=self.\_\_hostname, retained=True)).payload)

        return self.\_\_roomTemp

    def set\_roomTemp(self, temp): # Serverraumtemperatur in Cloud schreiben (macht eigentlich nur Sinn, wenn auf diesem Raspi die Temperaturmessung stattfindet)

        publish.single(self.\_\_topicRoomTemp, temp, hostname=self.\_\_hostname, retain=True)

        return

    def get\_setpointTemp(self): # Solltemperatur aus der Cloud lesen

        self.\_\_setpointTemp = float((subscribe.simple(self.\_\_topicSetpointTemp, hostname=self.\_\_hostname, retained=True)).payload)

        return self.\_\_setpointTemp

    def set\_setpointTemp(self, setpointTemp): # Solltemperatur in die Cloud schreiben

        publish.single(self.\_\_topicSetpointTemp, setpointTemp, hostname=self.\_\_hostname, retain=True)

        return

    def get\_hyst(self): # Hysterese aus der Cloud lesen

        self.\_\_hyst = float((subscribe.simple(self.\_\_topicHyst, hostname=self.\_\_hostname, retained=True)).payload)

        return self.\_\_hyst

    def set\_hyst(self, hyst): # Hysterese in die Cloud schreiben

        publish.single(self.\_\_topicHyst, hyst, hostname=self.\_\_hostname, retain=True)

        return

    def get\_gpioFan(self):

        self.\_\_gpioFan = int((subscribe.simple(self.\_\_topicGpioFan, hostname=self.\_\_hostname, retained=True)).payload)

        return self.\_\_gpioFan

    def set\_gpioFan(self, gpioFan): # GPIO Kanal für Ventilator in die Cloud schreiben

        publish.single(self.\_\_topicGpioFan, gpioFan, hostname=self.\_\_hostname, retain=True)

        return

    def update(self): # Hier steht erstmal nur die Raumtemp., es können aber weitere Parameter zur Aktualisierung eingebaut werden

        temp = self.\_\_tempsen.get\_celsius() # Auslesen der aktuellen Temperatur aus dem Sensor

        self.\_\_roomTemp = temp # Im Objektattribut speichern

        self.set\_roomTemp(temp) # Im Cloudspeicher aktualisieren

        return

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ende class Cloud

def main():