/\* --Code for ESP32 used in my Binda weather station \*/

#include <time.h>

#include <Wire.h>

#include <WiFi.h>

#include "FS.h"

#include "SPIFFS.h"

#include "BluetoothSerial.h"

#include <DS3231M.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_INA260.h>

#include <Adafruit\_BME280.h>

/\* Full API documentation is available here: https://pubsubclient.knolleary.net

https://github.com/knolleary/pubsubclient/tree/v2.8 \*/

#include <PubSubClient.h>

/\* You only need to format SPIFFS the first time you run a

test or else use the SPIFFS plugin to create a partition

https://github.com/me-no-dev/arduino-esp32fs-plugin \*/

#define FORMAT\_SPIFFS\_IF\_FAILED true

/\* WIFI NETWORK \*/

#define BMAX 128

char wifi\_ssid[BMAX] = "<your wifi ssid>";

char wifi\_password[BMAX] = "<your wifi password>";

int8\_t WifiReset\_hour = 2; // hour of day (24hr format) when wifi dongle is restarted

boolean fWifiReset = true; // set to true when action completed

WiFiClient;

/\* MQTT CONNECTION \*/

#define MQTTB 8192 // MQTT buffer

#define NONPAY 21 // estimated size of max non-payload bytes in MQTT packet

char mqtt\_server[BMAX] = "<your mqtt server website>";

char mqtt\_port[BMAX] = "<your mqtt port number>";

char mqtt\_user[BMAX] = "<your mqtt user name>";

char mqtt\_password[BMAX] = "<your mqtt password>";

#define SLAVE "ESP32" // message sent from slave to master

#define MASTER "W10" // message sent from master to the slave we want to do stuff

#define SLAVE\_END "ESP32\_END" // message sent from slave to master signifying arrival of last data block

char bufr[MQTTB]; // large global utility buffer

char bufs[BMAX]; // small global utility buffer

PubSubClient client(wifiClient);

/\* NTP SERVER \*/

char ntpServer[BMAX] = "pool.ntp.org";

long gmtOffset\_sec = 9\*60\*60; // +9 hrs (for EST)

int daylightOffset\_sec = 3600; // +1hr for Sydney winter

struct tm tms;

time\_t tma, dct;

int minGap=1; // gap in minutes between data collection events (assume 60 % minGap == 0)

char time\_minGap[BMAX]="1"; // string version of above (stored in config file)

int hourGap=24;

/\* ISR GENERAL \*/

volatile long debouncing\_time = 20; // milliseconds

volatile unsigned long time\_now;

#define DEL1 8000

#define DEL2 300

/\* ANEMOMETER \*/

#define WIND\_GPIO 19

const int nWIND=5;

volatile unsigned long last\_WIND;

volatile unsigned long aWIND[nWIND]; // array to store millis time stamps for cup anemometer spins

/\* RAIN GAUGE \*/

#define RAIN\_GPIO 18

const int nRAIN=100;

volatile unsigned long last\_RAIN;

volatile unsigned long aRAIN[nRAIN]; // array to store millis time stamps for rain gauge bucket tips

/\* WIND VANE \*/

#define VANE\_GPIO 34 // ADC1\_CH6

/\* RELAY \*/

#define RELAY\_GPIO 25

int8\_t From\_hour = 8; // hour of day (24hr format) from which charging of dongle is STOPPED

int8\_t To\_hour = 20; // hour of day (24hr format) from which charging of dongle is RESTARTED

/\* SERVO \*/

#define SERVO\_GPIO 26

int dutyCycle = 20; // experimental range [9=0deg, 32=180deg]

const int PWMFreq = 50;

const int PWMChannel = 0;

const int PWMResolution = 8;

boolean fPress = false;

time\_t dct2; // do action time

/\* BLUETOOTH SERVER \*/

BluetoothSerial SerialBT;

/\* I2C DEVICES/SENSORS \*/

Adafruit\_BME280 bme; // all in one weather sensor

Adafruit\_INA260 ina = Adafruit\_INA260(); // measures current and voltage

DS3231M\_Class rtc; // battery backed clock (RTC)

/\* misc global variables \*/

String Amsg = ""; // utility global string

int8\_t Now\_hour, Now\_min;

int8\_t ESP32Reboot\_hour = 1; // hour of day (24hr format) when this ESP32 is rebooted (hardcoded to 1AM)

boolean fESP32Reboot = true; // set to true when action completed

int8\_t Init\_hour = 23; // hour when action flags (fWifiReset & fESP32Reboot) are set to false

// ISR for anemometer

void IRAM\_ATTR WIND\_ISR() {

if ((unsigned long)(millis() - last\_WIND) > debouncing\_time) {

last\_WIND = millis();

// push last timestamp onto the end of the aWIND[] array (element nWIND-1). aWIND[0] is lost.

for (int j=1; j<nWIND; j++) aWIND[j-1] = aWIND[j];

aWIND[nWIND-1] = last\_WIND;

}

}

// ISR for rain gauge

void IRAM\_ATTR RAIN\_ISR() {

if ((unsigned long)(millis() - last\_RAIN) > debouncing\_time) {

last\_RAIN = millis();

// push last timestamp onto the end of the aRAIN[] array (element nRAIN-1). aRAIN[0] is lost.

for (int j=1; j<nRAIN; j++) aRAIN[j-1] = aRAIN[j];

aRAIN[nRAIN-1] = last\_RAIN;

}

}

// returns current windspeed in km/h

float getWindSpeed() {

if ((aWIND[0]==0) || (millis()-aWIND[nWIND-1] > 10000.0)) {

// Set wind speed to zero until nWIND time stamps have been collected

// or last time stamp occured more than 10 seconds ago.

return 0.0;

}

else {

// wind speed (in km/hr) is a calibration factor of 1200.0 divided

// the average of last nWIND-1 trigger time gaps (in milliseconds)

return 1200.0\*(float)(nWIND-1)/(float)(aWIND[nWIND-1] - aWIND[0]);

}

}

// returns mm of rain during last fh hours

float getCumulativeRainfall(float fh) {

float ts = millis(); // current time stamp as float

float tsh = ts - fh\*1000.0\*60.0\*60.0; // time stamp "fh" hours ago

if (tsh<0.0) tsh = 0.0; // no rain and initialization adjustment ;-)

// count number of times bucket has tipped during last "fh" hours

int wr = 0;

for (int j=nRAIN-1; j>=0; j--) { if (aRAIN[j]>tsh) wr++; }

// each bucket tip corresponds to 0.2794 mm of rain

return (float)wr\*0.2794;

}

// returns wind direction in degrees

float getWindDirection() {

// the iwd to x mapping was manually calibrated on the actual circuit

// with input to the VANE\_GPIO tapped below a 10k resistor.

float x;

int iwd = analogRead(VANE\_GPIO);

if ( 75 > iwd ) { x = 292.5; } // WNW

else if ( 150 > iwd ) { x = 247.5; } // WSW

else if ( 250 > iwd ) { x = 270.0; } // W

else if ( 400 > iwd ) { x = 337.5; } // NNW

else if ( 600 > iwd ) { x = 315.0; } // NW

else if ( 825 > iwd ) { x = 22.5; } // NNE

else if ( 1150 > iwd ) { x = 0.0; } // N

else if ( 1475 > iwd ) { x = 202.5; } // SSW

else if ( 1850 > iwd ) { x = 225.0; } // SW

else if ( 2185 > iwd ) { x = 67.5; } // ENE

else if ( 2375 > iwd ) { x = 45.0; } // NE

else if ( 2675 > iwd ) { x = 157.5; } // SSE

else if ( 2950 > iwd ) { x = 180.0; } // S

else if ( 3250 > iwd ) { x = 112.5; } // ESE

else if ( 3600 > iwd ) { x = 135.0; } // SE

else if ( 4096 > iwd ) { x = 90.0; } // E

else { x = -1.0; } // read error

return x;

}

// used to test BME280 sensor

void print\_BME280values() {

Serial.print("Temperature = "); Serial.print(bme.readTemperature()); Serial.println(" \xC2\xB0" "C");

Serial.print("Pressure = "); Serial.print(bme.readPressure()/100.0F); Serial.println(" hPa");

Serial.print("Humidity = "); Serial.print(bme.readHumidity()); Serial.println(" %");

Serial.println();

}

// used to test INA260 sensor

void print\_INA260values() {

Serial.print("Current: "); Serial.print(ina.readCurrent()); Serial.println(" mA");

Serial.print("Bus Voltage: "); Serial.print(ina.readBusVoltage()); Serial.println(" mV");

Serial.print("Power: "); Serial.print(ina.readPower()); Serial.println(" mW");

Serial.println();

}

// pFlag == 0 => read all sensors and publish for MASTER to see

// pFlag == 1 => read all sensors and append to a SPIFFS internal file

// pFlag == 2 => read all sensors and publish to bluetooth client

void readAllSensors(int pFlag) {

// read current, bus voltage and power from INA260 sensor into float variables

// do this first to avoid spurious

float rc = ina.readCurrent(); // in milliamps

float rv = ina.readBusVoltage()/1000.0F; // in Volts

float rw = ina.readPower(); // rw = (r)ead milli(w)atts

// read temperature, pressure and humidity from BME280 sensor into float variables

float rt = bme.readTemperature(); // in Celsius

float rp = bme.readPressure()/100.0F; // in HectoPascals

float rh = bme.readHumidity(); // in %

// read rainfall, windspeed and wind direction into float variables

float rr = getCumulativeRainfall(1.0); // during last 1 hour

float rs = getWindSpeed(); // in km/hr

float rd = getWindDirection(); // in degrees

// RTC temperature in C

float r3 = rtc.temperature()/100.0F;

// put these sensor readings into a space efficient comma delimited char array in bufa

int i, j=0, k, n; char buf[10][10], bufa[10\*10+35], buft[30];

// put global time variable tms into start of bufa (size depends on pFlag)

if (pFlag==0) {k=22; strftime(bufa,100,"%d-%b-%Y %H:%M:%S",&tms);} // include seconds

else {k=22-3; strftime(bufa,100,"%d-%b-%Y %H:%M",&tms);} // ignore seconds

strcpy(buft,bufa);

dtostrf(rt,8,2,buf[0]); // temperature as string (8 chars + 0 at end => 9 chars from buffer)

dtostrf(rp,8,1,buf[1]); // pressure as string

dtostrf(rh,8,1,buf[2]); // humidity as string

dtostrf(rr,8,2,buf[3]); // rainfall as string

dtostrf(rs,8,2,buf[4]); // wind speed as string

dtostrf(rd,8,1,buf[5]); // wind direction as string

dtostrf(r3,8,2,buf[6]); // RTC temp as string

dtostrf(rc,8,1,buf[7]); // current as string

dtostrf(rv,8,2,buf[8]); // bus voltage as string

dtostrf(rw,8,1,buf[9]); // power (in milliwatts) as string

bufa[k-2] = 44; bufa[k-1] = 32;

for(n=0;n<9;n++){for(i=0;i<8;i++){buf[n][i]==32 ? j++ :bufa[k+10\*n+i-j]=buf[n][i];} bufa[k+10\*n+8-j]=44; bufa[k+10\*n+9-j]=32;}

for(i=0;i<8;i++){buf[n][i]==32 ? j++ :bufa[k+10\*n+i-j]=buf[n][i];} bufa[k+10\*n+8-j]=10; bufa[k+10\*n+9-j]=0;

// Serial.print(bufa);

String sOut;

switch (pFlag) {

case 0: // publish for MASTER to see

// bufa[k+10\*9+8-j] = 0; // don't want to display '\n'

//client.publish(SLAVE, bufa, false);

sOut = "Timestamp "; sOut.concat(buft); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Temperature "; sOut.concat(buf[0]); sOut.concat(" C"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Pressure "; sOut.concat(buf[1]); sOut.concat(" hPa"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Humidity "; sOut.concat(buf[2]); sOut.concat(" %"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Rainfall "; sOut.concat(buf[3]); sOut.concat(" mm"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Wind speed "; sOut.concat(buf[4]); sOut.concat(" km/h"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Wind dir "; sOut.concat(buf[5]); sOut.concat(" deg"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "RTC temp "; sOut.concat(buf[6]); sOut.concat(" C"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Current "; sOut.concat(buf[7]); sOut.concat(" mA"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Voltage "; sOut.concat(buf[8]); sOut.concat(" V"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "Power "; sOut.concat(buf[9]); sOut.concat(" mW"); sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1: // append to SensorData.csv file

appendFileBasic(SPIFFS, "/SensorData.csv", bufa); // append line with '\n'

break;

case 2: // publish for Bluetooth client to see, append line with '\n'

SerialBT.print(bufa);

break;

}

}

// read current time into the tm structure using the battery backed RTC

void getLocalTimeRTC(tm \* t) {

// get the date and time from the RTC into a DateTime structure

DateTime dt = rtc.now();

// convert dt to a tm structure

t->tm\_year = dt.year()-1900; t->tm\_mon = dt.month()-1; t->tm\_mday = dt.day();

t->tm\_hour = dt.hour(); t->tm\_min = dt.minute(); t->tm\_sec = dt.second();

}

// Returns time in arithmetic representation (in global var dct) which is the nearest

// integral minute gap (in minGap global var) ahead of the current time (tms, tma) global vars.

// Assumes 60%minGag=0. eg. if minutes of now time is 23 and minGap=10 then returns time with min=40.

void createDataCollectionTime() {

getLocalTimeRTC(&tms);

tma = mktime(&tms); // convert to global time\_t variable

// rounds the minutes of current time down to lowest multiple of minGap

int gMin = (tms.tm\_min/minGap)\*minGap;

// adjusts the current time structure accordingly

struct tm tms2;

tms2.tm\_sec = 0; tms2.tm\_min = gMin; tms2.tm\_hour = tms.tm\_hour;

tms2.tm\_mday = tms.tm\_mday; tms2.tm\_mon = tms.tm\_mon; tms2.tm\_year = tms.tm\_year;

tms2.tm\_wday = tms.tm\_wday; tms2.tm\_yday = tms.tm\_yday; tms2.tm\_isdst = tms.tm\_isdst;

// converts this adjusted time structure to arithmetic representation & increases it by minGap

dct = mktime(&tms2) + minGap\*60;

// display tms2 and dct

char bufx[30]; strftime(bufx,100,"%d-%b-%Y %H:%M:%S",&tms2);

Serial.print("tms2= "); Serial.print(bufx); Serial.print(" dct= "); Serial.println(dct);

}

// Move the servo to position i in the range [8=0deg, 32=180deg] inclusive.

// Update the global position dutyCycle. Returns true if all ok, false otherwise.

// mFlag = 1 => move to position in increments, mFlag = 0 => move in one go.

boolean MoveServo(int i, int mFlag) {

int n;

if((i>=8)&&(i<=32)) {

if (mFlag==1) { // "slow" servo movement

if (i>dutyCycle) {

for (n=dutyCycle+1; n<=i; n++) {ledcWrite(PWMChannel,n);delay(15);}

dutyCycle = n-1;

} else if (i<dutyCycle) {

for (n=dutyCycle-1; n>=i; n--) {ledcWrite(PWMChannel,n);delay(15);}

dutyCycle = n+1;

}

} else if (mFlag==0) { // "fast" servo movement

ledcWrite(PWMChannel,i);

dutyCycle = i;

delay(15);

}

return true; // all ok

}

return false; // parse error

}

// presses the wifi dongle button for isec seconds

void wifi\_press(int isec) {

MoveServo(24, 0); // press button

time\_now = millis(); while (millis() < time\_now+isec\*1000) { } // wait for isec

MoveServo(20, 0); // back to neutral position

}

// switches wifi dongle off and then sets things up to switch it back on in 2 minutes

void wifi\_switchoffandon() {

wifi\_press(10); // presses wifi dongle button for 10 seconds (to shut it down)

getLocalTimeRTC(&tms); // read current time into global tm structure variable tms

dct2 = mktime(&tms)+(time\_t)(60\*2); // and convert to global time\_t variable dct2

fPress = true; // setup flag for doing restarting wifi dongle in 2 minutes

}

// MQTT callback routine. Driven by client.loop();

void callback(char\* topic, byte \*payload, unsigned int length) {

Serial.print("channel: "); Serial.print(topic);

Serial.print(" data: "); Serial.write(payload, length);

Serial.print(" length: "); Serial.print(length);

Serial.println();

/\* RESPOND TO PAYLOAD \*/

// converts payload to both String and char[] for easier processing ???

int i; String sOut;

Amsg = "";

for (i=0; i<length; i++) { Amsg += (char)payload[i]; }

for (i=0; i<length; i++) { bufr[i]=(char)payload[i]; } bufr[i]=0;

if (Amsg.equals("r")) {

// restarts ESP32

client.publish(SLAVE, "RESTARTING SLAVE", false);

Serial.println("Restarting in 5 seconds"); delay(5000);

ESP.restart();

} else if (Amsg.substring(0,1).equals("s")) {

// move servo to a position in the range [8=0deg, 32=180deg] inclusive

Amsg.remove(0,1); Amsg.trim();

if (MoveServo(Amsg.toInt(),0)) {

sOut = "Servo moved to position "; sOut.concat(itoa(dutyCycle,bufs,10));

sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

} else {

client.publish(SLAVE, "\*\*\* SERVO NOT MOVED - INVALID RANGE \*\*\*", false);

}

} else if (Amsg.equals("0")||Amsg.equals("1")) {

// Controls relay 0=off, 1=on

client.publish(SLAVE, "SET RELAY", false);

if (Amsg[0]=='0') { digitalWrite(25, LOW); }

if (Amsg[0]=='1') { digitalWrite(25, HIGH); }

} else if (Amsg.equals("m")) {

// Measure all sensors and publish with timestamp for MASTER to see

client.publish(SLAVE, "ALL SENSORS MEASURED", false);

readAllSensors(0); // 0 arguments sends to MASTER

} else if (Amsg.equals("d")) {

if (deleteFile(SPIFFS, "/SensorData.csv")) {

Serial.println("DELETED file SensorData.csv");

client.publish(SLAVE, "DELETED file SensorData.csv", false);

} else {

Serial.println("FAILED to delete file SensorData.csv");

client.publish(SLAVE, "FAILED to delete file SensorData.csv", false);

}

} else if (Amsg.equals("o")) {

// read/output SensorData.csv file

client.publish(SLAVE, "Displaying SensorData.csv", false);

if (readFile(SPIFFS, "/SensorData.csv")) {

client.publish(SLAVE, "Nothing to display - File is empty", false);

}

} else if (Amsg.substring(0,4).equals("wifi")) {

// read new WIFI credentials, send feedback to mqtt master.

// and if valid save credentials to SPIFFS file WIFIconfig.txt.

Amsg += (char)32;

parseWIFIcredentials(0);

} else if (Amsg.substring(0,4).equals("mqtt")) {

// read new MQTT credentials, send feedback to mqtt master,

// and if valid save credentials to SPIFFS file MQTTconfig.txt.

Amsg += (char)32; // a fudge?

parseMQTTcredentials(0);

} else if (Amsg.substring(0,4).equals("time")) {

// read new TIME credentials, send feedback to mqtt master,

// and if valid save credentials to SPIFFS file TIMEconfig.txt.

Amsg += (char)32; // a fudge?

parseTIMEcredentials(0);

} else if (Amsg.substring(0,2).equals("gd")) {

// Copy sFileName from SLAVE to MASTER in blocks

// Amsg is assumed to be "gd <aFilename>"

String sFileName = Amsg.substring(2); sFileName.trim();

sendFileBlocks(SPIFFS, sFileName);

} else if (Amsg.substring(0,4).equals("help")) {

// display help text

MQTThelp();

} else if (Amsg.equals("wr")) {

// (re)starting wifi dongle (by pressing for 5 seconds). If already on this does nothing!

wifi\_press(5);

sOut = "(RE)STARTED WIFI DONGLE"; sOut.toCharArray(bufr,MQTTB);

client.publish(SLAVE, bufr, false);

} else if (Amsg.equals("ws")) {

// shut down wifi dongle (by pressing for 10 seconds), wait for 2 minutes and then

// restart wifi dongle (by pressing for 5 secons).

sOut = "SHUT DOWN WIFI DONGLE FOR 2 MINUTES"; sOut.toCharArray(bufr,MQTTB);

client.publish(SLAVE, bufr, false);

wifi\_switchoffandon();

} else {

// no valid message received

sOut = "INVALID MESSAGE : "; sOut += bufr;

sOut.toCharArray(bufr,MQTTB);

client.publish(SLAVE, bufr, false);

}

}

// (Re)connect to WIFI

void connect\_WIFI() {

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

Serial.print(F("\n(Re)connecting to WIFI ")); Serial.print(wifi\_ssid); Serial.print(".");

WiFi.begin(wifi\_ssid, wifi\_password);

unsigned long tn = millis();

while (WiFi.status() != WL\_CONNECTED) {

unsigned long tn2 = millis();

while (millis() < tn2+DEL2) {};

if (millis() > tn+DEL1) break;

Serial.print(".");

}

if (WiFi.status() == WL\_CONNECTED) {

Serial.print("connected. IP address: "); Serial.println(WiFi.localIP());

}

else {

Serial.println("failed to connect :-(");

}

}

// Connects to MQTT broker and subscribes to selected topics. Reconnects to WiFi if necessary!

// bReconnect=true flag reconnects regardless if connection is ok.

void connect\_MQTT\_WIFI(bool bReconnect) {

unsigned long tn = millis();

// disconnect MQTT client if bReconnect flag is true

if (bReconnect) { client.disconnect(); delay(1000); };

// Loop until we are (re)connected to MQTT

while (!client.connected()) {

if (millis() > tn+DEL1) break;

Serial.print("(Re)connecting to MQTT...");

// Attempt to connect. set the 1st arg clientID to <mqtt\_user>.

client.setServer(mqtt\_server, atoi(mqtt\_port));

client.setCallback(callback);

Serial.print("\nbuffer size: before=");Serial.print(client.getBufferSize());

client.setBufferSize(MQTTB);

Serial.print(" after=");Serial.println(client.getBufferSize());

if (client.connect(mqtt\_user, mqtt\_user, mqtt\_password)) {

// Once connected, publish an announcement... and resubscribe

Serial.println("connected");

client.publish(SLAVE, "Slave (re)connected to MQTT broker", false);

client.subscribe(MASTER);

}

else {

// Failed to connect to MQTT

Serial.print("failed, rc="); Serial.print(client.state());

// WIFI might also be the cause.

connect\_WIFI();

}

}

}

void listDir(fs::FS &fs, const char \* dirname, uint8\_t levels){

Serial.printf("Listing directory: %s\r\n", dirname);

File root = fs.open(dirname);

if(!root){

Serial.println("- failed to open directory");

return;

}

if(!root.isDirectory()){

Serial.println(" - not a directory");

return;

}

File file = root.openNextFile();

while(file){

if(file.isDirectory()){

Serial.print(" DIR : "); Serial.println(file.name());

if(levels) listDir(fs, file.name(), levels -1);

} else {

Serial.print(" FILE: "); Serial.print(file.name());

Serial.print("\tSIZE: "); Serial.println(file.size());

}

file = root.openNextFile();

}

}

boolean readFile(fs::FS &fs, const char \* path){

char c; int i=0;

File file = fs.open(path);

if(!file || file.isDirectory()){

Serial.println("- failed to open file for reading");

return file;

}

Serial.printf("Reading file: %s Size: %d\n", path, file.size());

while(file.available()){

if ((c=file.read())!=10) {bufr[i]=c; i++;}

else {bufr[i]=0; Serial.println(bufr); client.publish(SLAVE, bufr, false); i=0;}

}

Serial.println("");

file.close();

return file;

}

// send an arbitrary SPIFFS file (if it exists) in blocks to the MASTER

// NOTE: assume file contains no bytes which are 0.

void sendFileBlocks(fs::FS &fs, String sFileName){

char apath[BMAX], c; int i=0, n=0; String sOut;

// create path to file by appending "/" to start of sFileName

sOut = "/"; sOut.concat(sFileName); sOut.toCharArray(apath,BMAX);

Serial.println(apath);

// try to open sFileName (using apath)

File file = fs.open(apath);

if(!file || file.isDirectory()){

sOut = "- failed to open file for reading"; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

Serial.println("- failed to open file for reading");

return;

}

// send "start combining blocks" message to MQTT master

// if consists of the string "gd <sFileName>"1

sOut = "gd "; sOut.concat(sFileName); sOut.concat(" "); sOut.concat(itoa(file.size(),bufs,10));

sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

Serial.printf("Reading file: %s Size: %d\n", apath, file.size()); Serial.println(bufs);

// reading individual characters, consolidating them into MQTTB-NONPAY-1 size blocks

// (excluding 0) and publishing these to MASTER

// NOTE: assume file does not contain the char with ascii value 0.

while(file.available()){

c=file.read();

if (i<(MQTTB-NONPAY-2)) {

bufr[i]=c; i++;

}

else {

bufr[i]=c; bufr[i+1]=0; // so number of chars excluding final 0 is MQTTB-NONPAY-1

client.publish(SLAVE, bufr, false);

i=0; n++;

}

}

// deal with last incomplete block (if any)

if ((i!=0)&&(i<(MQTTB-NONPAY-2))) {

bufr[i] = 0; n++;

client.publish(SLAVE, bufr, false);

}

file.close();

// send finished combining blocks message to MQTT master

Serial.print(n);Serial.println(" blocks sent to MASTER");

sOut = "Finished creating file"; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE\_END, bufs, false);

}

// reads the <wfif\_\*> global configuration parameters from their SPIFFS file.

void readWIFIconfig(fs::FS &fs, const char \* path){

int n=0, i=0, next; char c;

File file = fs.open(path);

while ((next = file.read()) != -1) {

c = (char)next;

switch (n) {

case 0: if (c==10) {wifi\_ssid[i]=0; i=0; n=1;} else {wifi\_ssid[i]=c; i++;}; break;

case 1: if (c==10) {wifi\_password[i]=0;} else {wifi\_password[i]=c; i++;}; break;

}

}

file.close();

// print out configuration

Serial.print("wifi ssid="); Serial.println(wifi\_ssid);

Serial.print("wifi password="); Serial.println(wifi\_password);

}

// reads the <mqtt\_\*> global configuration parameters from their SPIFFS file.

void readMQTTconfig(fs::FS &fs, const char \* path){

int n=0, i=0, next; char c;

File file = fs.open(path);

while ((next = file.read()) != -1) {

c = (char)next;

switch (n) {

case 0: if (c==10) {mqtt\_server[i]=0; i=0; n=1;} else {mqtt\_server[i]=c; i++;}; break;

case 1: if (c==10) {mqtt\_port[i]=0; i=0; n=2;} else {mqtt\_port[i]=c; i++;}; break;

case 2: if (c==10) {mqtt\_user[i]=0; i=0; n=3;} else {mqtt\_user[i]=c; i++;}; break;

case 3: if (c==10) {mqtt\_password[i]=0;} else {mqtt\_password[i]=c; i++;}; break;

}

}

file.close();

// print out configuration

Serial.print("mqtt server="); Serial.println(mqtt\_server);

Serial.print("mqtt port="); Serial.println(mqtt\_port);

Serial.print("mqtt user="); Serial.println(mqtt\_user);

Serial.print("mqtt password="); Serial.println(mqtt\_password);

}

// reads the <wfif\_\*> global configuration parameters from their SPIFFS file.

void readTIMEconfig(fs::FS &fs, const char \* path){

int n=0, i=0, next; char c;

File file = fs.open(path);

while ((next = file.read()) != -1) {

c = (char)next;

switch (n) {

case 0: if (c==10) {time\_minGap[i]=0;} else {time\_minGap[i]=c; i++;}; break;

}

}

file.close();

minGap = atoi(time\_minGap); // create int version of string

// print out configuration

Serial.print("time minGap="); Serial.println(time\_minGap);

}

// Basic wrapper. Returns error in boolean

bool deleteFile(fs::FS &fs, const char \* path){

return fs.remove(path);

}

// Basic wrapper. No messages or error checking.

void deleteFileBasic(fs::FS &fs, const char \* path){

fs.remove(path);

}

// Basic wrapper. No messages or error checking.

void appendFileBasic(fs::FS &fs, const char \* path, char \* message){

File file = fs.open(path, FILE\_APPEND);

file.print(message);

file.close();

}

// If parsing ok puts new WIFI credentials into the global variables <wifi\_\*> and

// saves to SPIFFS file WIFIconfig.txt for reloading at setup.

// iTo=1 => feedback to Bluetrack, iTo=0 => feedback to MQTT

void parseWIFIcredentials(int iTo) {

char wifi\_ssidTemp[BMAX], wifi\_passwordTemp[BMAX];

bool iParsed = false, iMaxed = false;

int m, k, i=0; char c, d; String sOut;

// loop through Amsg and populate the two <WIFI\_\*> global vars (0 terminated)

int n = Amsg.length()-1;

Amsg[n] = 32;

if (Amsg[4]==32) {

k = 0; m = 4;

while (m<=n) {

d = Amsg[m]; m++; c = Amsg[m];

if (!((c==32)&&(d==32))) {

switch (k) {

case 0:

if (c==32) { wifi\_ssidTemp[i]=0; if (i==BMAX) {k=2; iMaxed=true;} else {i=0; k=1;}; }

else { wifi\_ssidTemp[i]=c; i++;}; break;

case 1:

if (c==32) { wifi\_passwordTemp[i]=0; k=2; iMaxed=(i==BMAX); }

else { wifi\_passwordTemp[i]=c; i++;}; break;

}

}

}

// determine if parsing worked

if ((k==2)&&(!iMaxed)) iParsed=true;

}

if (iParsed) {

// now that parsing was ok can copy parsed strings to WIFI credentials

k=0; do { c = wifi\_ssidTemp[k]; wifi\_ssid[k] = c; k++; } while (c!=0);

k=0; do { c = wifi\_passwordTemp[k]; wifi\_password[k] = c; k++; } while (c!=0);

// depending on iTo flag

switch (iTo) {

case 0:

// send feedback to MQTT master... nicely formatted multi-messages

sOut = "VALID PARSE with new parameters:"; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "ssid= "; sOut += wifi\_ssid; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "password= "; sOut += wifi\_password; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// send feedback to Bluetooth client...

SerialBT.print("valid parse: new ssid="); SerialBT.print(wifi\_ssid);

SerialBT.print(" password="); SerialBT.println(wifi\_password);

break;

}

// restart WIFI with new credentials

connect\_WIFI();

// save credentials to SPIFFS file MQTTconfig.txt

char cLF[2] = "\n";

deleteFileBasic(SPIFFS, "/WIFIconfig.txt");

appendFileBasic(SPIFFS, "/WIFIconfig.txt", wifi\_ssid); appendFileBasic(SPIFFS, "/WIFIconfig.txt", cLF);

appendFileBasic(SPIFFS, "/WIFIconfig.txt", wifi\_password); appendFileBasic(SPIFFS, "/WIFIconfig.txt", cLF);

}

else {

// send feedback depending on iTo flag

switch (iTo) {

case 0:

// to MQTT master... nicely formatted multi-messages

(n==4) ? sOut = "Current parameters are:" : sOut = "CANNOT PARSE. Current parameters are:";

sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "ssid= "; sOut += wifi\_ssid; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "password= "; sOut += wifi\_password; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// On parsing failure send appropriate feedback to Bluetooth client

(n==4) ? SerialBT.print("current ssid=") : SerialBT.print("cannot parse: unchanged ssid=");

SerialBT.print(wifi\_ssid); SerialBT.print(" password="); SerialBT.println(wifi\_password);

break;

}

}

}

// If parsing ok puts new MQTT credentials into the global variables <mqtt\_\*> and

// saves to SPIFFS file MQTTconfig.txt for reloading at setup.

// iTo=1 => feedback to Bluetrack, iTo=0 => feedback to MQTT

void parseMQTTcredentials(int iTo) {

char mqtt\_serverTemp[BMAX], mqtt\_portTemp[BMAX], mqtt\_userTemp[BMAX], mqtt\_passwordTemp[BMAX];

bool iParsed = false, iMaxed = false;

int m, k, i=0; char c, d; String sOut;

// loop through Amsg and populate the two <MQTT\_\*> global vars

int n = Amsg.length()-1;

Amsg[n] = 32;

if (Amsg[4]==32) {

k = 0; m = 4;

while (m<=n) {

d = Amsg[m]; m++; c = Amsg[m];

if (!((c==32)&&(d==32))) {

switch (k) {

case 0:

if (c==32) { mqtt\_serverTemp[i]=0; if (i==BMAX) {k=4; iMaxed=true;} else {i=0; k=1;}; }

else { mqtt\_serverTemp[i]=c; i++;}; break;

case 1:

if (c==32) { mqtt\_portTemp[i]=0; if (i==BMAX) {k=4; iMaxed=true;} else {i=0; k=2;}; }

else { mqtt\_portTemp[i]=c; i++;}; break;

case 2:

if (c==32) { mqtt\_userTemp[i]=0; if (i==BMAX) {k=4; iMaxed=true;} else {i=0; k=3;}; }

else { mqtt\_userTemp[i]=c; i++;}; break;

case 3:

if (c==32) { mqtt\_passwordTemp[i]=0; k=4; iMaxed=(i==BMAX); }

else { mqtt\_passwordTemp[i]=c; i++;}; break;

}

}

}

// determine if parsing worked

if ((k==4)&&(!iMaxed)) iParsed=true;

}

if (iParsed) {

// now that parsing was ok can copy parsed strings to MQTT credentials

k=0; do { c = mqtt\_serverTemp[k]; mqtt\_server[k] = c; k++; } while (c!=0);

k=0; do { c = mqtt\_portTemp[k]; mqtt\_port[k] = c; k++; } while (c!=0);

k=0; do { c = mqtt\_userTemp[k]; mqtt\_user[k] = c; k++; } while (c!=0);

k=0; do { c = mqtt\_passwordTemp[k]; mqtt\_password[k] = c; k++; } while (c!=0);

// send feedback depending on iTo flag

switch (iTo) {

case 0:

// to MQTT master... nicely formatted multi-messages

sOut = "VALID PARSE with new parameters:"; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "server= "; sOut += mqtt\_server; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "port= "; sOut += mqtt\_port; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "user= "; sOut += mqtt\_user; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "password= "; sOut += mqtt\_password; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// to Bluetooth client...

SerialBT.print("valid parse: new server="); SerialBT.print(mqtt\_server);

SerialBT.print(" port="); SerialBT.print(mqtt\_port);

SerialBT.print(" user="); SerialBT.print(mqtt\_user);

SerialBT.print(" password="); SerialBT.println(mqtt\_password);

break;

}

// restart MQTT with new credentials

connect\_MQTT\_WIFI(true);

// save credentials to SPIFFS file MQTTconfig.txt

char cLF[2] = "\n";

deleteFileBasic(SPIFFS, "/MQTTconfig.txt");

appendFileBasic(SPIFFS, "/MQTTconfig.txt", mqtt\_server); appendFileBasic(SPIFFS, "/MQTTconfig.txt", cLF);

appendFileBasic(SPIFFS, "/MQTTconfig.txt", mqtt\_port); appendFileBasic(SPIFFS, "/MQTTconfig.txt", cLF);

appendFileBasic(SPIFFS, "/MQTTconfig.txt", mqtt\_user); appendFileBasic(SPIFFS, "/MQTTconfig.txt", cLF);

appendFileBasic(SPIFFS, "/MQTTconfig.txt", mqtt\_password); appendFileBasic(SPIFFS, "/MQTTconfig.txt", cLF);

}

else {

// on parsing failure send feedback depending on iTo flag

switch (iTo) {

case 0:

// to MQTT master... nicely formatted multi-messages

(n==4) ? sOut = "Current parameters are:" : sOut = "CANNOT PARSE. Current parameters are:";

sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "server= "; sOut += mqtt\_server; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "port= "; sOut += mqtt\_port; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "user= "; sOut += mqtt\_user; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "password= "; sOut += mqtt\_password; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// to Bluetooth client...

(n==4) ? SerialBT.print("current server=") : SerialBT.print("cannot parse: unchanged server=");

SerialBT.println(mqtt\_server);

SerialBT.print(" port="); SerialBT.print(mqtt\_port);

SerialBT.print(" user="); SerialBT.print(mqtt\_user);

SerialBT.print(" password="); SerialBT.println(mqtt\_password);

break;

}

}

}

// If parsing ok puts new TIME credentials into the global variables <time\_minGap> & <minGap> and

// saves to SPIFFS file TIMEconfig.txt for reloading at setup.

// iTo=1 => feedback to Bluetrack, iTo=0 => feedback to MQTT

void parseTIMEcredentials(int iTo) {

char minGapTemp[BMAX]; int iminGap;

bool iParsed = false, iMaxed = false;

int m, k, i=0; char c, d; String sOut;

// loop through Amsg and populate the <minGaTemp> var

int n = Amsg.length()-1;

Amsg[n] = 32;

if (Amsg[4]==32) {

k = 0; m = 4;

while (m<=n) {

d = Amsg[m]; m++; c = Amsg[m];

if (!((c==32)&&(d==32))) {

switch (k) {

case 0:

if (c==32) {minGapTemp[i]=0; k=1; iMaxed=(i==BMAX);}

else {minGapTemp[i]=c; i++;}; break;

}

}

}

// check that (int)minGapTemp is not <=0 and completely divides 60.

// If not force parse error by setting k=0.

iminGap = atoi(minGapTemp);

if (iminGap<=0) { k=0; }

else if (60%iminGap!=0) { k=0; }

else { itoa(iminGap,minGapTemp,10); } // base 10

// determine if parsing worked

if ((k==1)&&(!iMaxed)) iParsed=true;

}

if (iParsed) {

// now that parsing was ok can copy int and string global values to TIME credentials

minGap = iminGap;

k=0; do { c = minGapTemp[k]; time\_minGap[k] = c; k++; } while (c!=0);

// send feedback depending on iTo flag

switch (iTo) {

case 0:

// to MQTT master... nicely formatted multi-messages

sOut = "VALID PARSE with new parameter:"; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "time gap (mins)= "; sOut += time\_minGap; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// to Bluetooth client...

SerialBT.print("valid parse: new minGap="); SerialBT.println(time\_minGap);

break;

}

// this sets the next data collection time dct to the next minGap interval. eg if current time

// is 13min and minGap=10 then dtc=20min

createDataCollectionTime();

// Save credentials to SPIFFS file TIMEconfig.txt

char cLF[2] = "\n";

deleteFileBasic(SPIFFS, "/TIMEconfig.txt");

appendFileBasic(SPIFFS, "/TIMEconfig.txt", time\_minGap); appendFileBasic(SPIFFS, "/TIMEconfig.txt", cLF);

}

else {

// on parsing failure send feedback depending on iTo flag

switch (iTo) {

case 0:

// to MQTT master... nicely formatted multi-messages

(n==4) ? sOut = "Current parameter is:" : sOut = "CANNOT PARSE. Current parameter is:";

sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

sOut = "time gap (mins)= "; sOut += time\_minGap; sOut.toCharArray(bufs,BMAX);

client.publish(SLAVE, bufs, false);

break;

case 1:

// to Bluetooth client...

(n==4) ? SerialBT.print("current minGap=") : SerialBT.print("cannot parse: unchanged minGap=");

SerialBT.println(time\_minGap);

break;

}

}

}

// displays help on commands available via MQTT interface

void MQTThelp() {

client.publish(SLAVE, " r // restart this ESP32 in 5 seconds", false);

client.publish(SLAVE, " m // measure all sensors and display with time stamp", false);

client.publish(SLAVE, " d // delete SensorData.csv file", false);

client.publish(SLAVE, " o // display SensorData.csv file", false);

client.publish(SLAVE, " wifi [<ssid> <pwd>] // display or change slave wifi settings", false);

client.publish(SLAVE, " mqtt [<server> <port> <user> <pwd>] // display or change slave mqtt settings", false);

client.publish(SLAVE, " Xmqtt [<server> <port> <user> <pwd>] // display or change master mqtt settings", false);

client.publish(SLAVE, " time [<minGap>] // display or change data collection settings", false);

client.publish(SLAVE, " gd <filename> // Copy filename from slave to master", false);

client.publish(SLAVE, " help // display this help", false);

client.publish(SLAVE, " wr // (re)start wifi dongle", false);

client.publish(SLAVE, " ws // shut down wifi dongle and restart 2min later", false);

}

// displays help on commands available via Bluetooth interface

void BThelp() {

SerialBT.println("r // restart this ESP32 in 5 seconds");

SerialBT.println("m // measure all sensors and display with time stamp");

SerialBT.println("wifi [<ssid> <pwd>] // display or change slave wifi settings");

SerialBT.println("mqtt [<server> <port> <user> <pwd>] // display or change slave mqtt settings");

SerialBT.println("time [<minGap>] // display or change data collection settings");

SerialBT.println("help // display this help");

SerialBT.println("wr // (re)start wifi dongle");

SerialBT.println("ws // shut down wifi dongle and restart 2min later");

}

// THE Arduino IDE setup routine

void setup() {

// setup serial used for development/debugging

Serial.begin(115200); Serial.setTimeout(500);

// Start the I2C interface

Wire.begin();

/\* Setup and attach the LED PWM Channel to the GPIO Pin \*/

// setup LED PWM Channel for servo (Tower Pro SG90)

// this has a 180 deg range with duty cycle ra

ledcSetup(PWMChannel, PWMFreq, PWMResolution);

ledcAttachPin(SERVO\_GPIO, PWMChannel);

ledcWrite(PWMChannel, dutyCycle);

// SETUP SPIFFS and display base dir

if(!SPIFFS.begin(FORMAT\_SPIFFS\_IF\_FAILED)){

Serial.println("SPIFFS Mount Failed");

return;

}

listDir(SPIFFS, "/", 0);

// Now can read the WIFI & MQTT credentials from their SPIFFS files into the

// global variables <wifi\_\*>, <mqtt\_\*> <time\_\*> and respectively.

readWIFIconfig(SPIFFS, "/WIFIconfig.txt");

readMQTTconfig(SPIFFS, "/MQTTconfig.txt");

readTIMEconfig(SPIFFS, "/TIMEconfig.txt");

// Set WiFi to station mode and disconnect from an AP if it was previously connected

WiFi.mode(WIFI\_STA);

WiFi.disconnect(); delay(100);

Serial.println("\nWIFI Setup done");

// list the visible WIFI networks

Serial.print("Scanning for WIFI networks...");

int n = WiFi.scanNetworks();

Serial.println("completed");

if (n == 0) {

Serial.println("no networks found");

}

else {

Serial.print(n); Serial.println(" networks found");

for (int i = 0; i < n; ++i) {

// Print SSID and RSSI for each network found

Serial.print(i + 1); Serial.print(": ");

Serial.print(WiFi.SSID(i)); Serial.print(" (");

Serial.print(WiFi.RSSI(i)); Serial.print(")");

Serial.println((WiFi.encryptionType(i) == WIFI\_AUTH\_OPEN)?" ":"\*"); delay(10);

}

}

Serial.println(""); delay(500);

// Connect to WI-Fi and MQTT broker

connect\_MQTT\_WIFI(false);

// setup BME280 sensor

bool stat = bme.begin(0x77);

Serial.println(F("\nBME280 setup and test"));

if (!stat) { Serial.println("Could not find a valid BME280 sensor, check wiring!"); }

print\_BME280values();

// setup INA260 sensor

stat = ina.begin(0x40);

Serial.println(F("INA260 setup and test"));

if (!stat) { Serial.println("Could not find a valid INA260 sensor, check wiring!"); }

print\_INA260values();

// setup DS3231M rtc

while (!rtc.begin()) { Serial.println(F("Unable to find DS3231M. Checking again in 3s.")); delay(3000); }

Serial.println(F("DS3231M initialized."));

// connect to time server and update the date/time on the RTC

configTime(gmtOffset\_sec, daylightOffset\_sec, ntpServer); delay(2000);

getLocalTime(&tms,5000); // uses time server

if (tms.tm\_year!=70) { // internet working

// so can read time from time server and update RTC

rtc.adjust(DateTime(tms.tm\_year-100, tms.tm\_mon+1, tms.tm\_mday, tms.tm\_hour, tms.tm\_min, tms.tm\_sec));

}

Serial.print(F("DS3231M chip temperature is "));

Serial.print(rtc.temperature()/100.0F,2); // Value is in 100ths of a degree

Serial.println("\xC2\xB0" "C"); // print out degree character!

// update global time\_t variable dct for initial use in loop)

createDataCollectionTime();

// for testing - print out time in nice format

char bufx[30]; strftime(bufx,100,"%d-%b-%Y %H:%M:%S",&tms); Serial.println(bufx);

// connect to Bluetooth

bool btOK = SerialBT.begin("ESP32"); // This sets the Bluetooth device name

if (!btOK) { Serial.println("An error occurred initializing Bluetooth"); }

else { Serial.println("The device started, now you can pair it with bluetooth!"); }

// setup Wind speed and Rain gauge sensors. setup GPIO pins for interrupts.

pinMode(WIND\_GPIO, INPUT); attachInterrupt(WIND\_GPIO, WIND\_ISR, FALLING);

pinMode(RAIN\_GPIO, INPUT); attachInterrupt(RAIN\_GPIO, RAIN\_ISR, FALLING);

// setup GPIO pin for relay

pinMode(RELAY\_GPIO, OUTPUT); digitalWrite(RELAY\_GPIO, LOW);

}

// THE Arduino IDE loop routine

void loop() {

// the MQTT and WIFI connection loop

connect\_MQTT\_WIFI(false);

client.loop();

// Read current time using rtc (not relying on time server!)

getLocalTimeRTC(&tms); // read current time into global tm structure variable tms

tma = mktime(&tms); // and convert to global time\_t variable tma

Now\_hour = tms.tm\_hour; // extract hour for later use

Now\_min = tms.tm\_min; // extract minute for later use

// the READ ALL SENSORS (and append to data file) loop

if (tma > dct) {

Serial.println("Reading Sensors");

SerialBT.println("Reading Sensors");

// current time has just has just passed the data collection time.

// so collect and save sensor data (with tms as time-stamp)

readAllSensors(1);

dct += minGap\*60; // increment/update data collection time

// the WIFI DONGLE CHARGING loop where not charging in [From\_hour, To\_hour) range

// inside READ ALL SENSORS loop to avoid continuous running digitalWrite all the time

if ((From\_hour <= Now\_hour)&&(Now\_hour < To\_hour)) {

digitalWrite(25, HIGH); // relay on = not charging wifi dongle

}

else {

digitalWrite(25, LOW); // relay off = charging wifi dongle

}

}

// the RESTART WIFI DONGLE loop (after it was shut down earlier)

if (fPress) {

if (tma > dct2) {

wifi\_press(5); // restarts wifi dongle

fPress = false; // since restart completed can reset fPress flag

Serial.println("Restarting wifi dongle");

SerialBT.println("Restarting wifi dongle");

}

}

// the RESET WIFI DONGLE loop (at a fixed time daily, here 0AM)

// and 2 minutes later restarted in previous loop!

if ((Now\_hour==WifiReset\_hour)&&(!fWifiReset)) {

Serial.println("Resetting wifi dongle");

SerialBT.println("Resetting wifi dongle");

fWifiReset = true; // so we don't reset more than once during this hour

wifi\_switchoffandon();

}

// the REBOOT ESP32 loop (at a fixed time daily, here 1AM)

if ((Now\_hour==ESP32Reboot\_hour)&&(!fESP32Reboot)) {

// fESP32Reboot = true; // set so that rebooting is not repeated - not needed since lost on restart

Serial.println("Restarting ESP32");

SerialBT.println("Restarting ESP32");

delay(1000); ESP.restart();

}

// the RESET ACTIONS loop (at a fixed time daily, here 23AM)

if ((Now\_hour==Init\_hour)&&(Now\_min==0)) {

Serial.println("Resetting Actions");

SerialBT.println("Resetting Actions");

fWifiReset = false; // so that can reset wifi at next action time (WifiReset\_hour)

fESP32Reboot = false; // so that can reboot ESP32 at next action time (ESP32Reboot\_hour)

}

// the BLUETOOTH loop

// messages from Bluetooth terminal are assumed to end with CR=13 & LF=10.

// Read until LF found. CR is then replaced by 0 to make null terminating string

// this string is then parsed and acted upon.

if (SerialBT.available() > 0) {

Amsg = SerialBT.readStringUntil(10); Amsg.replace(13,0);

Serial.println(Amsg);

if(Amsg.equals("r")) {

SerialBT.println("Restarting slave in 5 seconds");

Serial.println("Restarting slave in 5 seconds");

delay(5000); ESP.restart();

}

else if(Amsg.equals("m")) {

// collect sensor data with current time stamp 'tms'

// and print to Bluetooth terminal

readAllSensors(2);

}

else if(Amsg.substring(0,4).equals("wifi")) {

// read new WIFI credentials, send feedback to Bluetooth client,

// and if valid save credentials to SPIFFS file WIFIconfig.txt.

parseWIFIcredentials(1);

}

else if(Amsg.substring(0,4).equals("mqtt")) {

// read new MQTT credentials, send feedback to Bluetooth client,

// and if valid save credentials to SPIFFS file MQTTconfig.txt.

parseMQTTcredentials(1);

}

else if(Amsg.substring(0,4).equals("time")) {

// read new TIME credentials, send feedback to Bluetooth client,

// and if valid save credentials to SPIFFS file TIMEconfig.txt.

parseTIMEcredentials(1);

}

else if(Amsg.equals("help")) {

BThelp();

}

else if (Amsg.equals("wr")) {

// (re)starting wifi dongle (by pressing for 5 seconds). If already on this does nothing!

SerialBT.println("(RE)STARTED WIFI DONGLE");

wifi\_press(5);

}

else if (Amsg.equals("ws")) {

// shut down wifi dongle (by pressing for 10 seconds), wait for 2 minutes and then

// restart wifi dongle (by pressing for 5 seconds).

SerialBT.println("SHUT DOWN WIFI DONGLE FOR 2 MINUTES");

wifi\_switchoffandon();

}

else {

SerialBT.println("Sent invalid message");

}

}

/\* JUST HERE FOR FUN

time\_now = millis();

while (millis() < time\_now+1000) { }

\*/

}