Jaime Huenchulao, Alex Tardón

Practica profecional 2 | LAb Cimm ubb

Documentación:

Codigo del esp32

Índice

[Introducción 2](#_Toc125485393)

[Ensamblaje del dispositivo 3](#_Toc125485394)

[Comunicación Aplicación – Dispositivo 4](#_Toc125485395)

[Almacenamiento 5](#_Toc125485396)

[Código del ESP32 6](#_Toc125485397)

# Introducción

El presente documento tiene por finalidad esclarecer, detallar y especificar, las funciones del código del ESP32 y uso en conjunto con la aplicación de Android.

Este proyecto se basa en la tesis de otros alumnos, los cuales crearon un dispositivo usando un esp32 y un encoder óptico para medir la deformación que se producen en paredes, el objetivo del proyecto asignado por el Profesor Luis Vera, es crear una aplicación móvil que permita al usuario interactuar con el dispositivo sin la necesidad de modificar el código cada vez que se necesitaba cambiar una variable, así como también agregar funciones al proyecto, por ejemplo la opción de crear un agendamiento para dejar funcionando el dispositivo durante un periodo especifico de tiempo, la opción de dejar un temporizador y que la app muestre los datos en tiempo real, entre otras funciones.

En ese sentido y tras analizarlo, determinamos que el proyecto requería una forma de comunicación entre el ESP32 y la aplicación, para ello utilizamos comunicación serial mediante bluetooth y una serie de mensajes con un formato claro para que el dispositivo ejecutase las funciones solicitadas, además la aplicación espera devuelta ciertos valores para validar los efectos de estas.

# Ensamblaje del dispositivo

Antes de ahondar en el ámbito del código y lógica del dispositivo (ESP32), debemos ser capaces de reproducir el mismo, este cuenta con una placa ESP32 de Espressif Systems, específicamente el modelo de desarrollo DEV KIT V1 en conjunto con una placa datalogger con un RTC DS1307 y un slot de SD, además del antes mencionado encoder óptico.

Se debe montar la placa del datalogger sobre la del DEV KIT asegurándonos de que los pines coincidan, en cuanto el encoder, este debe conectarse de la siguiente forma:

* El cable verde se conecta al pin 7
* El cable blanco se conecta al pin 6
* El cable rojo se conecta al pin de 5V
* El cable negro se conecta al GND

A continuación, se presentan una serie de fotografías de referencia:

# Comunicación Aplicación – Dispositivo

Como se mencionó en la introducción, se utiliza la comunicación serial mediante bluetooth, específicamente la librería “BluetoothSerial.h” (se recomienda revisar la documentación del proyecto original para instalar las librerías en el IDE de Arduino). Entre la aplicación y el dispositivo se transmiten mensajes, estos mensajes no son nada más que Strings con formatos específicos los cuales son recibidos y iterados por el ESP32 para determinar que función debe ejecutar.

Por ejemplo si se quiere utilizar el temporizador, en la app se selecciona la duración, y al pulsar el botón de “Iniciar” la app manda el siguiente mensaje al ESP32, “TIMER; mm:ss:ms;” (con mm minutos, ss segundos y ms milisegundos). Cuando este mensaje llega al ESP32 entra en el “loop()” donde se obtiene el mensaje completo y se itera con strtok(&message[0],”;”) para obtener la primera parte (TIMER) y luego llama a la función funcTimer(), la cual obtiene los valores del timer, los transforma a milisegundos y activa el loop del booleano isTimer. Sin embargo, la app, requiere saber si se activó o no el temporizador, por lo que esta queda a la espera de recibir un 1, que indica que todo está en orden. Mientras el tiempo no se acabe el ESP32 manda los datos captados por el encoder óptico a la app y esta los muestra en el gráfico. Para finalizar el temporizador, la app, quien lleva a cabo el conteo del tiempo de manera independiente, envía un mensaje de “OFF;” al ESP32 y espera de vuelta un mensaje de confirmación de que se apagó correctamente (1). Tras ello, la app solicita devuelta el archivo donde se almacenan estos datos en el ESP32.

Lo descrito en el ejemplo anterior ocurre en gran parte de las funciones de la aplicación, sin embargo, no ocurre para todas, recomendamos que si se busca ahondar o modificar la comunicación entre la app y el ESP32 se utilice la aplicación putty y Serial Bluetooth Terminal (en Android) para ver las interacciones entre la app y el dispositivo.

Algunas consideraciones generales de la comunicación son las siguientes:

1. La app para validar que el mensaje fue entregado en su totalidad necesita recibir un \*, el \* indica que el mensaje esta completo, de lo contrario el mensaje será recibido por la app pero no será procesado (de igual forma el \* se elimina del resto del mensaje, solo se usa para confirmar que es el final del mensaje).
2. Debido a limitaciones de memoria del ESP32 los archivos no se pueden cargar completos y enviarlos en un solo mensaje (de ahí la necesidad del punto anterior), para esto se usa la función readBT() que manda los archivos por partes.
3. Los datos son transmitidos con un “!” al principio, lo que le permite a la app identificar el valor como un dato que se debe mostrar en el gráfico.
4. La función check() recibe el valor que retorna cada función y manda el código a la app.
5. Solo algunas funciones requieren recibir el valor de confirmación, por ejemplo SCAN;GET; no requiere de este.

# Almacenamiento

Para almacenar los datos recopilados por el ESP32 se utiliza una memoria SD en la cual se generan 2 archivos, “data.txt” y “schedule.txt”, ocasionalmente y de manera temporal, se puede generar un archivo “temp.txt” el cual se usa para escribir un nuevo archivo “schedule.txt” sin el agendamiento que se busca eliminar. En el archivo de “data.txt” los datos se encuentran almacenados con el siguiente formato (todos en una sola línea):

“START;Nombre;FechaDeInicio;TiempoDeMuestreo;UnidadDeMedida;!dato1!dato2!dato3…STOP;FechaDeTermino;”

La cantidad de “!dato” es variable y las fechas siguen el formato de YYYY/MM/DDTHH:MM:SS, el Tiempo de muestreo se encuentra en milisegundos, además, la variable de nombre es opcional.

En el archivo de “schedule.txt” los agendamientos se encuentran almacenados con el siguiente formato:

“Nombre;FechaDeInicio;FechaDeTermino-”

Y al igual que en “data.txt”, las fechas siguen el formato de YYY/MM/DDTHH:MM:SS.

# Código del ESP32

En cuanto al código, este se encuentra documentado, línea a línea casi en su totalidad, y cada función presenta sus parámetros y valores retornados, se recomienda visualizar el código en visual studio code, ya que facilita el acceso a la documentación de cada función con solo pasar el mouse sobre la función en cuestión.

*/\* 1.- LIBRERIAS Y VARIABLES\*/*

*// Librarias para RIC DS1307*

*#include* "RTClib.h"

*// Librarias para Bluetooth Serial (256 bytes de buffer)*

*#include* "BluetoothSerial.h"

*// Librerias para tarjeta SD*

*#include* "FS.h"

*#include* "SD.h"

*#include* <SPI.h>

*#define* SD\_CS 5 *// Se define el CS en el pin 5 y se nombra como SD\_CS*

BluetoothSerial SerialBT;

RTC\_DS1307 rtc;

*// Variables para el codificador rotativo*

int counter = 0; *// Esta variable aunentará o disminuirá dependiendo de la rotación del codificador*

int temp;

const float pi = 3.14159;

const float R = 1.905;

const int N = 1200;

float distance = 0;

const int BUFFER\_SIZE = 256; *// Tamaño del buffer para la comunicacion Bluetooth*

*/\**

*el formato para las configuraciones es:*

*"CONFIG;[nombre de la configuracion];[valor de la configuracion]"*

*\*/*

*// Varibles del experimiento*

int varTiempoMuestreo = 100;

*/\**

*el formato del mensaje para crear un agendamiento es:*

*"AGENDAR;stDate;edDate;stTime;edTime;"*

*\*/*

*// variables para el agendamiento*

String nSchedule;

DateTime nextSchedule, endNextSchedule;

bool enAgendamiento = false;

*// Variables de comunicacion*

String message;

char \*token;

*// variables para almacenar el tiempo del temporizador*

int timer = 0;

bool isTimer = false;

*// variables para el manejo del loop de on*

bool isOn = false;

*/\* 2.- SETUP\*/*

void setup()

{

*// asignando valores iniciales a las variables de agendamiento y fecha para evitar errores*

    nextSchedule = DateTime("9999:12:30", "10:10:10");

    endNextSchedule = DateTime("9999:12:30", "10:10:10");

    Serial.begin(115200);

    SerialBT.begin("ESP32Prueba"); *// Mombre del dispositivo Bluetooth*

    Serial.println("El dispositivo inicio, ¡ahora puedes emparejarlo con bluetooth!");

*if* (!rtc.begin())

    {

        Serial.println("No se pudo encontrar el RTC");

        Serial.flush();

    }

*if* (!rtc.isrunning())

    {

        Serial.println("RIC NO se está ejecutando, ¡configurenos la hora!");

*// Cuando es necesario configurar la hora en un muevo dispositivo, o despues de una perdida de energia,*

*// la siguiente linea establece el RIC en la fecha y hora que se compilo este codigo.*

        rtc.adjust(DateTime(F(\_\_DATE\_\_), F(\_\_TIME\_\_)));

    }

*// Inicializar tarjeta SD*

    SD.begin(SD\_CS);

*if* (!SD.begin(SD\_CS))

    {

        Serial.println("Falló el montaje de la tarjeta");

*return*;

    }

*// almacena el tipo de sd*

    uint8\_t cardType = SD.cardType();

*if* (cardType == CARD\_NONE)

    {

        Serial.println("Sin tarjeta SD adjunta");

*return*;

    }

    Serial.println("Inicializando tarjeta SD...");

*if* (!SD.begin(SD\_CS))

    {

        Serial.println("ERROR - ¡Falló la inicialización de la tarjeta 5D!");

*return*; *// init failed*

    }

*// Si el archivo data.txt no existe*

*// Cree un archivo en la tarjeta SD y escriba las etiquetas de datos*

    File file = SD.open("/data.txt");

*if* (!file)

    {

        Serial.println("El archivo no existe");

        Serial.println("Creando archivo. ..");

        writeFile(SD, "/data.txt", "Lectura de fecha, hora y contador");

    }

*else*

    {

        Serial.println("El archivo ya existe");

    }

    file.close();

    File nFile = SD.open("/schedules.txt");

*if* (!nFile)

    {

        Serial.println("El archivo no existe");

        Serial.println("Creando archivo. ..");

        writeFile(SD, "/schedules.txt", "");

    }

*else*

    {

        Serial.println("El archivo ya existe");

    }

    nFile.close();

    pinMode(27, INPUT\_PULLUP); *// Entrada PullUp interna pin 27*

    pinMode(14, INPUT\_PULLUP); *// Entrada PullUp interna pin 14*

*// Configuración de interrupción (atrachInterrupt)*

*// Balso ascendente de la fase A codificado activa a10()*

    attachInterrupt(14, ai0, RISING);

*// Ba1so ascendente de la fase B codificado activa ail()*

    attachInterrupt(27, ai1, RISING);

*// Comprobacion de interrupciones si hay error en los archivos de datos y agendamientos*

*// checkData() retorna 0 si no hay error y si hay error retorna la cantidad de errores*

    file = SD.open("data.txt");

*if* (checkData("data.txt") < 0)

    {

        Serial.print("error en datos");

    }

}

*/\* 3.- FUNCIONES DEL SETUP \*/*

void ai0()

{

*if* (digitalRead(27) == LOW)

    {

        counter++;

    }

*else*

    {

        counter--;

    }

}

void ai1()

{

*if* (digitalRead(14) == LOW)

    {

        counter--;

    }

*else*

    {

        counter++;

    }

}

*// funcion para comprobar archivos en caso de interrupcion no llamar*

*// @param fileName nombre del archivo a comprobar*

*// @return 0 si no hay error, -1 si hay error*

int checkData(String *fileName*)

{

    File file = SD.open("/" + *fileName*);

*// almacena el contenido del archivo en una variable*

    String fileContent = "";

*while* (file.available())

    {

        fileContent += (char)file.read();

*if* (fileContent.length() > 20)

        {

            fileContent.substring(1);

        }

    }

    Serial.print(fileContent);

    file.close();

    int searchPos = 0;

    int errors = 0;

*// busca los pares de START; y STOP; en el archivo*

    int end = fileContent.indexOf("STOP;", searchPos);

*// si no hay END; despues de START; hay un error*

*if* (end == -1)

    {

*// si hay un error escribir el stop en el archivo*

        saveToSDCard(*fileName*, "STOP;" + dateNow() + ";#");

        errors++;

*return* -1;

    }

*return* 1;

}

*/\* 4.- FUNCIONES DEL SD CARD \*/*

*// funcion para guardar datos en el archivo especificado en la tarjeta SD*

*// @param file nombre del archivo*

*// @param toSave datos a guardar*

*// @return void*

void saveToSDCard(String *file*, String *toSave*)

{

    appendFile(SD, ("/" + *file*).c\_str(), *toSave*.c\_str());

}

*// Escribir en la tarjeta SD*

*// @param fs tarjeta SD*

*// @param path ruta del archivo*

*// @param message mensaje a escribir*

*// @return void*

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

{

    Serial.printf("Escribiendo archivo: %s\n", path);

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

*if* (!file)

    {

        Serial.println("No se pudo abrir el archivo para escribir");

*return*;

    }

*if* (file.print(message))

    {

        Serial.println("Archivo escrito");

    }

*else*

    {

        Serial.println("Escritura fallida");

    }

    file.close();

}

*// Anexar datos a la tarjeta SD*

*// @param fs tarjeta SD*

*// @param path ruta del archivo*

*// @param message mensaje a escribir*

*// @return void*

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

{

    Serial.printf("Appending to file: %s\n", path);

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

*if* (!file)

    {

        Serial.println("Error al abrir el archivo para adjuntar");

*return*;

    }

*if* (file.print(message))

    {

        Serial.println("Mensaje adjunto");

    }

*else*

    {

        Serial.println("Adjuntar fallo");

    }

    file.close();

}

*// Funcion para enviar archivos por bluetooth serial pero considerando el tamaño del buffer (BUFFER\_SIZE)*

*// @param nFile nombre del archivo a leer*

*// @return 1 si no hay error, -1 si hay error*

int readBT(String *nFile*)

{

    File file = SD.open("/" + nFile);

*if* (!file)

    {

        Serial.println("El archivo no existe");

*return* -1;

    }

    int total = 0;

*while* (file.available())

    {

        String chunk = "";

        int byteRead = 0;

*while* (byteRead < BUFFER\_SIZE && file.available())

        {

            chunk += (char)file.read();

            byteRead++;

        }

        total += byteRead;

*if* (nFile.compareTo("schedule.txt"))

        {

            chunk.replace(',', ';');

            SerialBT.println(chunk);

        }

*else*

        {

            SerialBT.println(chunk);

        }

        Serial.println(chunk);

    }

*if* (total == 0)

    {

        SerialBT.print("1");

    }

    file.close();

    SerialBT.println("\*");

*// delay para la funcion de check*

    delay(100);

*return* 1;

}

*/\* 5.- FUNCIONES DE LA LOGICA DEL ESP32 \*/*

*// funcion para comprobar el resultado una funcion y mandar un mensaje por bluetooth*

*// para que la aplicacion de android lo reciba y sepa que hacer 1 = OK, 0 = No se realizo ninguna accion, -1 = Error*

*// @param code codigo que retorna la funcion*

*// @return void*

void check(int *code*)

{

*// esta seccion es para el debug por el puerto serie*

*if* (code == -1)

    {

        Serial.println("Error");

    }

*else* *if* (code == 1)

    {

        Serial.println("OK");

    }

*else* *if* (code == 0)

    {

        Serial.println("No se realizo ninguna accion");

    }

*// delay para dejar que la app reciba el primer mensaje*

    delay(1000);

*// siempre se manda un mensaje por bluetooth para indicar el resultado de la funcion*

*// la applicacion de android se encarga de interpretar el mensaje (formato: <codigo>\*)*

    SerialBT.println(String(code) + "\*");

    Serial.println(String(code)+"\*");

}

*// funcion para fortmatear la fecha y hora en un formato legible por el DateTime de la libreria RTClib*

*// @param void*

*// @return String fecha y hora en formato legible*

String dateNow()

{

    DateTime now = rtc.now();

*return* String(now.year(), DEC) + "/" + String(now.month(), DEC) + "/" + String(now.day(), DEC) + "T" + String(now.hour(), DEC) + ":" + String(now.minute(), DEC) + ":" + String(now.second(), DEC) + ":" + "0";

}

*// funcion para obtener la fecha y hora de un DateTime*

*// @param dat fecha y hora en formato DateTime*

*// @return String fecha y hora*

String getDateString(DateTime dat)

{

*return* String(dat.year(), DEC) + "/" + String(dat.month(), DEC) + "/" + String(dat.day(), DEC) + "T" + String(dat.hour(), DEC) + ":" + String(dat.minute(), DEC) + ":" + String(dat.second(), DEC) + ":" + "0";

}

*// funcion para cambiar los valores de las variables de configuracion (formato: "NOMBRE;VALOR")*

*// @param message mensaje con la configuracion a realizar*

*// @return 1 si se realizo la configuracion, 0 si no se encontro la configuracion, -1 si hubo un error*

int config(char \**message*)

{

*// se lee la configuracion a realizar y se almacena en configName*

    char \*configName = strtok(message, ";");

*// se lee el valor de la configuracion y se almacena en configValue*

    char \*configValue = strtok(NULL, ";");

*// se lee el valor de la configuracion y se asigna a la variable correspondiente*

*if* (strcmp(configName, "TIEMPOMUESTREO") == 0)

    {

*if* (atoi(configValue) >= 100)

        {

            varTiempoMuestreo = atoi(configValue);

*return* 1;

        }

*else*

        {

*// mensaje de error por tiempo de muestreo muy corto*

*return* -1;

        }

    }

*// si no se encuentra la configuracion se retorna 0*

*return* 0;

}

*// funcion que reinicia los valores de las variables de configuracion a los valores por defecto*

*// @param void*

*// @return 1 si se reinicio la configuracion, -1 si hubo un error*

int reset()

{

    varTiempoMuestreo = 100;

*return* 1;

}

*// funcion para crear agendamientos*

*// @param message mensaje con los datos del agendamiento*

*// @return 1 si se creo el agendamiento, -1 si hubo un error*

int createAgendamiento(char \**message*)

{

*// el formato de llegada del mensaje es: "Nombre,FechaInicio,FechaFin"*

*// con las fechas con formato: "YYYY/MM/DDTHH:MM:SS"*

*// se asigna el mensaje recibido a una variable String para poder usar la funcion substring*

    String sc = message;

*// se crean variables para almacenar los valores de la fecha y hora de inicio y fin del agendamiento y el nombre del agendamiento*

    String scStart = sc;

    String scEnd = sc;

    String nSchedule = sc;

*// se reemplazan los caracteres "/" por "-" para poder crear un objeto DateTime por ejemplo: // Prueba1,2023/1/18T18:10:0,2023/1/20T19:0:0*

    scStart.replace("/", "-"); *// Prueba1,2023-1-18T18:10:0,2023-1-20T19:0:0*

    scEnd.replace("/", "-");

*// se obtienen los valores de la fecha y hora de inicio y fin del agendamiento y el nombre del agendamiento (formato: "Nombre,FechaInicio,FechaFin")*

    scStart = scStart.substring(sc.indexOf(",") + 1, sc.length()); *// Prueba1,2023-1-18T18:10:0,2023-1-20T19:0:0*

    scEnd = scEnd.substring(sc.indexOf(",") + 1, sc.length());

*// se obtienen los valores de la fecha y hora de inicio y fin del agendamiento (formato: "FechaInicio,FechaFin" -> "YYYY-MM-DDTHH:MM:SS,YYYY-MM-DDTHH:MM:SS")*

    scStart = scStart.substring(0, scStart.indexOf(",") - 2); *// 2023-1-18T18:10:0*

    Serial.println("string start: " + scStart);

*// se obtienen los valores de la fecha y hora de fin del agendamiento (formato: "FechaFin" -> "YYYY-MM-DDTHH:MM:SS")*

    scEnd = scEnd.substring(scEnd.indexOf(",") + 1, scEnd.length() - 2); *// 2023-1-20T19:0:0*

    Serial.println("string end: " + scEnd);

    nSchedule = nSchedule.substring(0, sc.indexOf(",")); *// Prueba1*

*// se crea un objeto DateTime con los valores de la fecha y hora de inicio y fin del agendamiento*

    DateTime newSchedule(scStart.c\_str());

    DateTime newEndSchedule(scEnd.c\_str());

*// compara la fecha y hora de inicio del agendamiento con la fecha y hora de inicio del siguiente agendamiento*

*if* (nextSchedule > newSchedule)

    {

*// si la fecha y hora de inicio del agendamiento es menor a la fecha y hora de inicio del siguiente agendamiento*

*// se asigna la fecha y hora de inicio del agendamiento a la fecha y hora de inicio del siguiente agendamiento*

        nextSchedule = newSchedule;

        Serial.println(getDateString(newSchedule));

*// se asigna la fecha y hora de fin del agendamiento a la fecha y hora de fin del siguiente agendamiento*

        endNextSchedule = newEndSchedule;

    }

*// se crea un string con el formato "Nombre,FechaInicio,FechaFin-"*

    sc = sc + "-";

*// se guarda el string en el archivo "schedules.txt"*

    saveToSDCard("schedules.txt", sc);

    Serial.println("hora actual" + dateNow());

*return* 1;

}

*// ! esta funcion solo es valida si los agendamientos se crean en orden cronologico*

*// funcion para cargar agendamientos del archivo "schedules.txt"*

*// @param void*

*// @return 1 si se cargo el agendamiento, -1 si hubo un error*

int loadSchedule()

{

    File file = SD.open("schedules.txt");

*if* (!file)

    {

*return* -1;

    }

*// se crea una variable String para almacenar un schedule a la vez*

    String schedule = "";

*// se lee el archivo "schedules.txt" y se almacena en la variable String*

*while* (file.available())

    {

        schedule = file.readStringUntil('-');

*// se obtiene la fecha y hora de inicio del agendamiento*

        String scStart = schedule.substring(schedule.indexOf(",") + 1, schedule.length());

        scStart = scStart.substring(0, scStart.indexOf(",") - 2);

*// se obtiene la fecha y hora de fin del agendamiento*

        String scEnd = schedule.substring(schedule.indexOf(",") + 1, schedule.length());

        scEnd = scEnd.substring(scEnd.indexOf(",") + 1, scEnd.length() - 2);

*// se crea un objeto DateTime con los valores de la fecha y hora de inicio y fin del agendamiento*

        DateTime newSchedule(scStart.c\_str());

        DateTime newEndSchedule(scEnd.c\_str());

*// compara la fecha y hora de inicio del agendamiento con la fecha y hora de inicio del siguiente agendamiento*

*if* (nextSchedule > newSchedule)

        {

*// si la fecha y hora de inicio del agendamiento es menor a la fecha y hora de inicio del siguiente agendamiento*

*// se asigna la fecha y hora de inicio del agendamiento a la fecha y hora de inicio del siguiente agendamiento*

            nextSchedule = newSchedule;

*// se asigna la fecha y hora de fin del agendamiento a la fecha y hora de fin del siguiente agendamiento*

            endNextSchedule = newEndSchedule;

*// se asigna el nombre del agendamiento a la variable nameSchedule*

            nSchedule = schedule.substring(0, schedule.indexOf(","));

        }

    }

    file.close();

*return* 1;

}

*// funcion para encender el muestreo de datos, enviarlos por bluetooth y guardarlos en el archivo "data.txt"*

*// @param void*

*// @return void*

int on()

{

*if* (counter != temp)

    {

        distance = ((2 \* pi \* R) / N) \* counter;

        Serial.println(); *// creo que no es necessario por el println de abajo*

        temp = counter;

    }

*// Se envia el valor de la distancia al dispositivo conectado por bluetooth con el formato de "!distancia"*

    String d = "!" + String(distance, DEC);

    SerialBT.println(d + "\*"); *// ! el asterisco para que la reciba el mensaje la app*

    Serial.println(d);

*// se guarda el valor de la distancia en el archivo "data.txt"*

    saveToSDCard("data.txt", d);

}

*// funcion para manejar el temporizador*

*// @param message mensaje recibido por bluetooth con el formato "TIMER;TIEMPO;"*

*// @return 1 si se activo el temporizador, -1 si hubo un error*

int funcTimer(char \**message*)

{

*// iterar el mensaje para obtener el valor del tiempo formato: "TIMER;TIEMPO;"*

*// el TIEMPO viene en formato mm:ss:ms => 00:00:000*

*// se obtiene el valor de los minutos*

    char \*m = strtok(NULL, ":");

*// se obtiene el valor de los segundos*

    char \*s = strtok(NULL, ":");

*// se obtiene el valor de los milisegundos*

    char \*ms = strtok(NULL, ";");

*// se asigna el valor del tiempo en milisegundos al int timer*

    timer = (atoi(m) \* 60000) + (atoi(s) \* 1000) + atoi(ms);

*// se comprueba si el tiempo no es 0*

*if* (timer == 0)

    {

*return* -1;

    }

*// se activa el bool isTimer*

    isTimer = true;

*// se debe crear un string con el formato "START;;FECHAINICIO;TIEMPO;!DISTANCIA;FECHAFIN;STOP;"*

*// y se guarda en el archivo "data.txt"*

    String dat = "START;;" + dateNow() + ";" + String(varTiempoMuestreo, DEC) + ";" + "cm" + ";";

    saveToSDCard("data.txt", dat);

*return* 1;

}

*// funcion para elimianr un agendamiento*

*// @param message mensaje recibido por bluetooth con el formato "DELETE;NOMBRE;"*

*// @return 1 si se elimino el agendamiento, -1 si hubo un error, 0 si no se encontro el agendamiento*

int deleteSchedule(char \**message*)

{

*// se obtiene el nombre del agendamiento*

    char \*n = strtok(NULL, ";");

*// si el agendamiento a eliminar es el actual*

*if* (strcmp(nSchedule.c\_str(), n) == 0)

    {

*// isOn se asigna el valor de false para que no se ejecute el muestreo*

        isOn = false;

*// enAgendamiento se asigna el valor de false para que no se termine el muestreo si no se ha iniciado un agendamiento*

        enAgendamiento = false;

*// se crea un string con el formato "STOP;FechaHoraDeTermino"*

        String dat = "STOP;" + dateNow() + ";";

*// se guarda el string en el archivo "data.txt"*

        saveToSDCard("data.txt", dat);

*// se manda el data.txt por bluetooth al terminar el agendamiento*

*// readBT("data.txt");*

*// delay(100);*

*// se carga el siguiente agendamiento*

        loadSchedule();

*return* 1;

    }

*// se crea un objeto File con el archivo "schedules.txt"*

    File file = SD.open("/schedules.txt");

*// se crea un archivo temporal para guardar los agendamientos*

    File temp = SD.open("/temp.txt", FILE\_WRITE);

    bool found = false;

*// se lee el archivo "schedules.txt" hasta el final*

*while* (file.available())

    {

*// se lee una linea del archivo "schedules.txt"*

        String line = file.readStringUntil('-');

*if* (line.indexOf(n) == -1)

        {

            temp.print(line + "-");

        }

*else*

        {

            found = true;

        }

    }

*// se cierra el archivo "schedules.txt"*

    file.close();

*// se cierra el archivo temporal*

    temp.close();

*if* (!found)

    {

        SD.remove("/temp.txt");

*return* 0;

    }

*// si exite el agendamiento se elimina el archivo "schedules.txt"*

    SD.remove("/schedules.txt");

*// se renombra el archivo temporal como "schedules.txt"*

    SD.rename("/temp.txt", "/schedules.txt");

*// delay por si las dudas*

*return* 1;

}

*/\* 6.- LOOP \*/*

*// funcion que admistra el funcionamiento del dispositivo*

void loop()

{

*// si estamos en un temporizador se ejecuta el siguiente código*

*if* (isTimer)

    {

*// se llama al on() para encender el muestreo*

        on();

*// se espera el tiempo de muestreo*

        delay(varTiempoMuestreo);

*// se resta el tiempo de muestreo al tiempo restante del temporizador*

        timer = timer - varTiempoMuestreo;

*// si el tiempo restante del temporizador es menor o igual a 0 se ejecuta el siguiente código*

*if* (timer <= 0)

        {

            isTimer = false;

*// ! no hace falta guardar el stop porque la app manda un OFF; para terminar el muestreo*

        }

    }

*// se comprueba si el siguiente agendamiento es menor o igual a la fecha y hora actual (se inicia el agendamiento)*

*if* (nextSchedule <= rtc.now())

    {

*// se asigna la fecha y hora de inicio del siguiente agendamiento un valor muy alto para que no se cumpla la condicion*

        nextSchedule = DateTime("9999:12:30", "10:10:10");

*if*(isOn){

*// se crea un string con el formato "STOP;FechaHoraDeTermino"*

            String dat = "STOP;" + dateNow() + ";";

*// se guarda el string en el archivo "data.txt"*

            saveToSDCard("data.txt", dat);

            SerialBT.println("sAbort\*");

        delay(3000);

*// se manda el data.txt por bluetooth al terminar el agendamiento*

            readBT("data.txt");

            delay(3000);

    }

*// se crea un string con el formato "START;Nombre;FechaHoraDeInicio;TiempoDeMuestreo;UnidadDeMedida;"*

        String dat = "START;" + nSchedule + ";" + dateNow() + ";" + String(varTiempoMuestreo, DEC) + ";" + "cm" + ";";

*// se guarda el string en el archivo "data.txt" y se inicia el muestreo*

        saveToSDCard("data.txt", dat);

*// se asigna el valor de la variable isOn a true para que se inicie el muestreo*

        isOn = true;

*// guarda el inicio del agendamiento en data y empieza a tomar la data*

        enAgendamiento = true;

*// se manda un mensaje por bluetooth para indicar que se inicio el agendamiento (no estoy seguro sobre el c\_str())*

    SerialBT.println("scheduleStart\*");

    delay(1000);

    }

*// si la variable isOn es true se ejecuta el siguiente código*

*if* (isOn)

    {

*// se llama a la funcion on() para encender el muestreo*

        on();

*// se espera el tiempo de muestreo*

        delay(varTiempoMuestreo);

*// se comprueba si la fecha y hora de fin del agendamiento es menor o igual a la fecha y hora actual*

*// y si la variable enAgendamiento es true (para que no se termine el muestreo si no se ha iniciado un agendamiento)*

*if* (enAgendamiento && endNextSchedule <= rtc.now())

        {

*// isOn se asigna el valor de false para que no se ejecute el muestreo*

            isOn = false;

*// enAgendamiento se asigna el valor de false para que no se termine el muestreo si no se ha iniciado un agendamiento*

            enAgendamiento = false;

*// se crea un string con el formato "STOP;FechaHoraDeTermino"*

            String dat = "STOP;" + dateNow() + ";";

*// se guarda el string en el archivo "data.txt"*

            saveToSDCard("data.txt", dat);

        delay(100);

        SerialBT.println("scheduleStop\*");

        delay(1000);

        SerialBT.println("1\*");

        delay(1000);

*// se manda el data.txt por bluetooth al terminar el agendamiento*

            readBT("data.txt");

            delay(1000);

*// se carga el siguiente agendamiento*

            loadSchedule();

        }

    }

*else*

    {

        delay(100);

    }

*// Si se recibe un mensaje por Bluetooth (SerialBT) se ejecuta el siguiente código*

*if* (SerialBT.available())

    {

*// Se lee el mensaje y se almacena en la variable message*

        message = SerialBT.readString();

*// Se imprime el mensaje en el monitor serial*

*// Se divide el mensaje en un token*

        token = strtok(&message[0], ";");

*// Se compara el primer token con las palabras reservadas*

*if* (strcmp(token, "CONFIG") == 0)

        {

            check(config(&message[0]));

        }

*else* *if* (strcmp(token, "SCAN") == 0)

        {

            token = strtok(NULL, ";");

*if* (strcmp(token, "GET") == 0)

            { *// MANDAR DATA.txt*

                readBT("data.txt");

                delay(100);

            }

        }

*else* *if* (strcmp(token, "SCHEDULE") == 0)

        {

            token = strtok(NULL, ";");

*if* (strcmp(token, "ADD") == 0)

            {

                token = strtok(NULL, ";");

                check(createAgendamiento(token));

                delay(100);

            }

*else* *if* (strcmp(token, "DELETE") == 0)

            {

                check(deleteSchedule(token));

                delay(100);

            }

*else*

            { *// MANDAR SCHEDULES.txt*

                readBT("schedules.txt");

                delay(100);

            }

        }

*else* *if* (strcmp(token, "AGENDAR") == 0)

        {

            check(createAgendamiento(&message[0]));

        }

*else* *if* (strcmp(token, "ON") == 0)

        {

*// el formato de encendido en el archivo es START;;TiempoDeMuestreo;UnidadDeMedida;*

            isOn = true;

            String dat = "START;;" + dateNow() + ";" + String(varTiempoMuestreo, DEC) + ";" + "cm" + ";";

            saveToSDCard("data.txt", dat);

            SerialBT.println("1\*");

            delay(100);

        }

*else* *if* (strcmp(token, "TIMER") == 0)

        {

*// el formato de encendido en el archivo es START;;TiempoDeMuestreo;UnidadDeMedida;*

            check(funcTimer(&message[0]));

*// manda el archivo cuando se acaba el temporizador*

            delay(100);

        }

*else* *if* (strcmp(token, "OFF") == 0)

        {

            isOn = false;

            isTimer = false;

            String dat = "STOP;" + dateNow() + ";";

            saveToSDCard("data.txt", dat);

            SerialBT.println("1\*");

            delay(100);

*// se carga el siguiente agendamiento*

            loadSchedule();

        }

*else* *if* (strcmp(token, "READ") == 0)

        {

*// el formato de lectura es READ; y lee el archivo data.txt*

            check(readBT("data.txt"));

        }

*else* *if* (strcmp(token, "RESET") == 0)

        {

*// el formato de RESET es RESET;*

            check(reset());

        }

*else*

        {

            Serial.println("Error en el mensaje");

            SerialBT.print("-1\*"); *// PUEDE SER PRINTLN*

        }

*// Se limpia la variable message*

        message = "";

    }

}