MCGA

NODO EDGE

Andrés Felipe Fernández Ríos
Elkin Alejandro Ledesma Narvaez
Angie Tatiana Perez Muñoz

Índice de clases

Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

Índice de archivos

Lista de archivos

Lista de todos los archivos con breves descripciones:

Documentación de clases

Referencia de la estructura dht_message_t

Paquete de datos del sensor DHT (temp, hum), LDR y MQ5.

Atributos públicos

- float **temp**
- float **hum**
- int **luz**
- int **mq5**

Descripción detallada

Paquete de datos del sensor DHT (temp, hum), LDR y MQ5.

Definición en la línea **90** del archivo **NodoEdge.ino**.

Documentación de datos miembro

float dht_message_t::hum

Humedad relativa en %.

Definición en la línea **92** del archivo **NodoEdge.ino**.

int dht_message_t::luz

Lectura del sensor de luz (LDR).

Definición en la línea **93** del archivo **NodoEdge.ino**.

int dht_message_t::mq5

Lectura del sensor de gas MQ5.

Definición en la línea **94** del archivo **NodoEdge.ino**.

float dht_message_t::temp

Temperatura ambiental en °C.

Definición en la línea **91** del archivo **NodoEdge.ino**.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/Andres Fernandez/Desktop/ProyectoMGA/NodoEdge/**NodoEdge.ino**

Referencia de la estructura soil_message_t

Paquete de datos del sensor de humedad de suelo.

Atributos públicos

- float **soil**

Descripción detallada

Paquete de datos del sensor de humedad de suelo.

Definición en la línea **100** del archivo **NodoEdge.ino**.

Documentación de datos miembro

float soil_message_t::soil

Humedad de suelo en %.

Definición en la línea **101** del archivo **NodoEdge.ino**.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/Andres Fernandez/Desktop/ProyectoMGA/NodoEdge/**NodoEdge.ino**

Documentación de archivos

Referencia del archivo C:/Users/Andres Fernandez/Desktop/ProyectoMGA/NodoEdge/NodoEdge.ino

```
#include <WiFi.h>
#include <esp_now.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <time.h>
#include "esp_wifi.h"
#include "SD_MMC.h"
```

Clases

struct **dht_message_t***Paquete de datos del sensor DHT (temp, hum), LDR y MQ5.*

struct **soil_message_t***Paquete de datos del sensor de humedad de suelo.*

defines

- #define **BOTtoken** "7672713502:AAH55Vyi1rU-EF1eij3dnpqYvRMLxzsE58g"
  Token del bot de Telegram (obtenido de BotFather).

- #define **CHAT_ID** "5365006160"
  ID del chat o canal donde se enviarán los mensajes.

- #define **TEMP_THRESHOLD** 30.0
  Temperatura máxima permitida (°C).

- #define **HUM_THRESHOLD** 80.0
  Límite inferior de humedad de suelo (%).

- #define **SOIL_THRESHOLD_LOW** 30.0
  Límite superior de humedad de suelo (%).

- #define **SOIL_THRESHOLD_HIGH** 70.0
  Umbral de LDR: valores altos indican poca luz.

- #define **LDR_THRESHOLD** 700.0
  Umbral de MQ5: detección de gas/metano.

- #define **MQ5_THRESHOLD** 1500.0

Funciones

- UniversalTelegramBot **bot** (**BOTtoken**, **client**)
  Instancia de UniversalTelegramBot para envío de mensajes.

- void **saveDataToSD** (float temp, float hum, float soil, int luz, int mq5)
  Guarda una línea de datos en un archivo en SD_MMC.

- void **OnDataRecv** (const esp_now_recv_info_t *info, const uint8_t *incoming, int len)
  Maneja la llegada de paquetes ESP-NOW desde los nodos.

- void **DHTTask** (void *parameter)
  FreeRTOS Task que revisa la cola dhtQueue y genera alertas.

- void **SoilTask** (void *parameter)
  FreeRTOS Task que revisa soilQueue y genera alertas de suelo.

- void **TelegramTask** (void *parameter)
  FreeRTOS Task que revisa msgQueue y envía alertas a Telegram.

- void **setup** ()
  Función setup de Arduino.

- void **loop** ()
  Función loop de Arduino.

Variables

- const char * **ssid** = "iPhone"
  SSID y contraseña de la red WiFi a la que conectarse.

- const char * **password** = "25310tati"
- WiFiClientSecure **client**
  Cliente WiFi seguro para HTTPS.

- const char * **ntpServer** = "pool.ntp.org"
  Servidor NTP a utilizar para sincronizar fecha y hora.

- const long **gmtOffset_sec** = -5 * 3600
  Offset en segundos respecto a UTC (-5h para Colombia).

- const int **daylightOffset_sec** = 0
  Offset de horario de verano (en este caso 0).

- uint8_t **unifiedSenderMAC** [] = {0x08, 0xD1, 0xF9, 0xEE, 0x4F, 0x2C}
- uint8_t **soilSenderMAC** [] = {0xA0, 0xB7, 0x65, 0x1B, 0x18, 0x50}
- QueueHandle_t **dhtQueue**
- QueueHandle_t **soilQueue**
- QueueHandle_t **msgQueue**

Documentación de «define»

#define BOTtoken "7672713502:AAH55Vyi1rU-EF1eij3dnpqYvRMLxzsE58g"

Token del bot de Telegram (obtenido de BotFather).

Definición en la línea **34** del archivo **NodoEdge.ino**.

#define CHAT_ID "5365006160"

ID del chat o canal donde se enviarán los mensajes.
Definición en la línea **39** del archivo **NodoEdge.ino**.

#define HUM_THRESHOLD 80.0

Límite inferior de humedad de suelo (%).
Definición en la línea **72** del archivo **NodoEdge.ino**.

#define LDR_THRESHOLD 700.0

Umbral de MQ5: detección de gas/metano.
Definición en la línea **78** del archivo **NodoEdge.ino**.

#define MQ5_THRESHOLD 1500.0

Definición en la línea **80** del archivo **NodoEdge.ino**.

#define SOIL_THRESHOLD_HIGH 70.0

Umbral de LDR: valores altos indican poca luz.
Definición en la línea **76** del archivo **NodoEdge.ino**.

#define SOIL_THRESHOLD_LOW 30.0

Límite superior de humedad de suelo (%).
Definición en la línea **74** del archivo **NodoEdge.ino**.

#define TEMP_THRESHOLD 30.0

Temperatura máxima permitida (°C).
<
Humedad ambiental máxima (%).
Definición en la línea **70** del archivo **NodoEdge.ino**.

Documentación de funciones

UniversalTelegramBot bot (BOTtoken , client)

Instancia de UniversalTelegramBot para envío de mensajes.

void DHTTask (void * parameter)

FreeRTOS Task que revisa la cola dhtQueue y genera alertas.

Parámetros

parameter	No utilizado.

Crea un mensaje de alerta si algún valor supera su umbral y lo envía a msgQueue.

Definición en la línea **181** del archivo **NodoEdge.ino**.

void loop ()

Función loop de Arduino.

Nota

Vacío: la lógica corre en tareas y callbacks.

Definición en la línea **311** del archivo **NodoEdge.ino**.

void OnDataRecv (const esp_now_recv_info_t * info, const uint8_t * incoming, int len)

Maneja la llegada de paquetes ESP-NOW desde los nodos.

Parámetros

info	Estructura con información del origen.
incoming	Puntero a los datos recibidos.
len	Longitud de los datos.

Clasifica paquetes según MAC, extrae estructura y la envía a la cola correspondiente. También guarda cada medición en SD.

Definición en la línea **160** del archivo **NodoEdge.ino**.

void saveDataToSD (float temp, float hum, float soil, int luz, int mq5)

Guarda una línea de datos en un archivo en SD_MMC.

Parámetros

temp	Valor de temperatura (°C).
hum	Valor de humedad relativa (%).
soil	Valor de humedad de suelo (%).
luz	Lectura de LDR.
mq5	Lectura de MQ5.

Organiza los directorios como /YYYY-MM-DD/HH/data.txt, crea carpetas si no existen y añade una línea con los valores.

Definición en la línea **121** del archivo **NodoEdge.ino**.

void setup ()

Función setup de Arduino.

Conecta a WiFi, sincroniza NTP, monta SD, crea colas y tareas, configura ESP-NOW y registra callbacks.

Definición en la línea **251** del archivo **NodoEdge.ino**.

void SoilTask (void * parameter)

FreeRTOS Task que revisa soilQueue y genera alertas de suelo.

Parámetros

parameter	No utilizado.

Definición en la línea **207** del archivo **NodoEdge.ino**.

void TelegramTask (void * parameter)

FreeRTOS Task que revisa msgQueue y envía alertas a Telegram.

Parámetros

parameter	No utilizado.

Definición en la línea **233** del archivo **NodoEdge.ino**.

Documentación de variables

WiFiClientSecure client

Cliente WiFi seguro para HTTPS.

Definición en la línea **44** del archivo **NodoEdge.ino**.

const int daylightOffset_sec = 0

Offset de horario de verano (en este caso 0).

Definición en la línea **65** del archivo **NodoEdge.ino**.

QueueHandle_t dhtQueue

Cola para mensajes **dht_message_t**

Definición en la línea **105** del archivo **NodoEdge.ino**.

const long gmtOffset_sec = -5 * 3600

Offset en segundos respecto a UTC (-5h para Colombia).

Definición en la línea **60** del archivo **NodoEdge.ino**.

QueueHandle_t msgQueue

Cola para punteros a cadenas de alerta

Definición en la línea **107** del archivo **NodoEdge.ino**.

const char* ntpServer = "pool.ntp.org"

Servidor NTP a utilizar para sincronizar fecha y hora.

Definición en la línea **55** del archivo **NodoEdge.ino**.

const char* password = "25310tati"

Definición en la línea **28** del archivo **NodoEdge.ino**.

QueueHandle_t soilQueue

Cola para mensajes **soil_message_t**

Definición en la línea **106** del archivo **NodoEdge.ino**.

uint8_t soilSenderMAC[] = {0xA0, 0xB7, 0x65, 0x1B, 0x18, 0x50}

Nodo humedad de suelo

Definición en la línea **84** del archivo **NodoEdge.ino**.

const char* ssid = "iPhone"

SSID y contraseña de la red WiFi a la que conectarse.

Definición en la línea **27** del archivo **NodoEdge.ino**.

uint8_t unifiedSenderMAC[] = {0x08, 0xD1, 0xF9, 0xEE, 0x4F, 0x2C}

Nodo DHT+LDR+MQ5

Definición en la línea **83** del archivo **NodoEdge.ino**.

NodoEdge.ino

Ir a la documentación de este archivo.

```
00001
00014
00015 #include <WiFi.h>
00016 #include <esp_now.h>
00017 #include <WiFiClientSecure.h>
00018 #include <UniversalTelegramBot.h>
00019 #include <time.h>
00020 #include "esp_wifi.h"
00021 #include "SD_MMC.h"
00022
00023 // ── Configuración WiFi ──
00027 const char* ssid     = "iPhone";
00028 const char* password = "25310tati";
00029
00030 // ── Configuración del bot de Telegram ──
00034 #define BOTtoken  "7672713502:AAH55Vyi1rU-EF1eij3dnpqYvRMLxzsE58g"
00035
00039 #define CHAT_ID   "5365006160"
00040
00044 WiFiClientSecure client;
00045
00049 UniversalTelegramBot bot(BOTtoken, client);
00050
00051 // ── Configuración NTP ──
00055 const char* ntpServer = "pool.ntp.org";
00056
00060 const long gmtOffset_sec = -5 * 3600;
00061
00065 const int daylightOffset_sec = 0;
00066
00067 // ── Umbrales de alerta ──
00068
00070 #define TEMP_THRESHOLD      30.0
00072 #define HUM_THRESHOLD       80.0
00074 #define SOIL_THRESHOLD_LOW  30.0
00076 #define SOIL_THRESHOLD_HIGH 70.0
00078 #define LDR_THRESHOLD       700.0
00080 #define MQ5_THRESHOLD       1500.0
00081
00082 // ── Direcciones MAC de nodos emisores ──
00083 uint8_t unifiedSenderMAC[] = {0x08, 0xD1, 0xF9, 0xEE, 0x4F, 0x2C};
00084 uint8_t soilSenderMAC[]   = {0xA0, 0xB7, 0x65, 0x1B, 0x18, 0x50};
00085
00086 // ── Estructuras de datos recibidos ──
00090 typedef struct {
00091     float temp;
00092     float hum;
00093     int luz;
00094     int mq5;
00095 } dht_message_t;
00096
00100 typedef struct {
00101     float soil;
00102 } soil_message_t;
00103
00104 // ── Colas de FreeRTOS ──
00105 QueueHandle_t dhtQueue;
00106 QueueHandle_t soilQueue;
00107 QueueHandle_t msgQueue;
00108
00109 // ── Función para guardar datos en SD_MMC ──
00121 void saveDataToSD(float temp, float hum, float soil, int luz, int mq5) {
00122     struct tm timeinfo;
00123     if (!getLocalTime(&timeinfo)) {
00124         Serial.println("â□Œ Error al obtener la hora");
00125         return;
00126     }
00127
00128     char datePath[16];
00129     char hourPath[20];
```

```
00130     char filePath[28];
00131
00132     strftime(datePath, sizeof(datePath), "/%Y-%m-%d", &timeinfo);
00133     strftime(hourPath, sizeof(hourPath), "/%Y-%m-%d/%H", &timeinfo);
00134     strftime(filePath, sizeof(filePath), "/%Y-%m-%d/%H/data.txt", &timeinfo);
00135
00136     if (!SD_MMC.exists(datePath)) SD_MMC.mkdir(datePath);
00137     if (!SD_MMC.exists(hourPath)) SD_MMC.mkdir(hourPath);
00138
00139     File file = SD_MMC.open(filePath, FILE_APPEND);
00140     if (file) {
00141         file.printf("Temp: %.1f°C, Hum: %.1f%%, Suelo: %.1f%%, Luz: %d, MQ5: %d\n",
00142                     temp, hum, soil, luz, mq5);
00143         file.close();
00144         Serial.println(String("âœ… Datos guardados en ") + filePath);
00145     } else {
00146         Serial.println("â□Œ Error al abrir el archivo");
00147     }
00148 }
00149
00150 // ── Callback de recepción ESP-NOW ──
00160 void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t *incoming, int len)
{
00161     if (memcmp(info->src_addr, soilSenderMAC, 6) == 0 && len ==
sizeof(soil_message_t)) {
00162         soil_message_t s;
00163         memcpy(&s, incoming, len);
00164         xQueueSend(soilQueue, &s, portMAX_DELAY);
00165         saveDataToSD(0, 0, s.soil, 0, 0);
00166     } else if (memcmp(info->src_addr, unifiedSenderMAC, 6) == 0 && len ==
sizeof(dht_message_t)) {
00167         dht_message_t d;
00168         memcpy(&d, incoming, len);
00169         xQueueSend(dhtQueue, &d, portMAX_DELAY);
00170         saveDataToSD(d.temp, d.hum, 0, d.luz, d.mq5);
00171     }
00172 }
00173
00174 // ── Tarea para procesar datos DHT/LDR/MQ5 ──
00181 void DHTTask(void *parameter) {
00182     dht_message_t d;
00183     for (;;) {
00184         if (xQueueReceive(dhtQueue, &d, portMAX_DELAY)) {
00185             String msg = "";
00186             Serial.printf("ðŸŒ¡ Temp=%.1fÂ°C ðŸ’§ Hum=%.1f%% ðŸ”† Luz=%d ðŸ”¥
MQ5=%d\n", d.temp, d.hum, d.luz, d.mq5);
00187
00188             if (d.temp > TEMP_THRESHOLD)      msg += String("âš  Alta temperatura:
") + d.temp + "Â°C\n";
00189             if (d.hum > HUM_THRESHOLD)        msg += String("ðŸ’§ Alta humedad: ")
+ d.hum + "%\n";
00190             if (d.luz > LDR_THRESHOLD)        msg += String("ðŸŒ‘ Luz baja: ") +
d.luz + "\n";
00191             if (d.mq5 > MQ5_THRESHOLD)        msg += String("ðŸ”¥ Posible gas
detectado: ") + d.mq5 + "\n";
00192
00193             if (msg.length()) {
00194                 char* buf = (char*)malloc(msg.length() + 1);
00195                 msg.toCharArray(buf, msg.length() + 1);
00196                 xQueueSend(msgQueue, &buf, portMAX_DELAY);
00197             }
00198         }
00199     }
00200 }
00201
00202 // ── Tarea para procesar datos de humedad de suelo ──
00207 void SoilTask(void *parameter) {
00208     soil_message_t s;
00209     for (;;) {
00210         if (xQueueReceive(soilQueue, &s, portMAX_DELAY)) {
00211             String msg = "";
00212             Serial.printf("ðŸŒ± Humedad del suelo: %.1f%%\n", s.soil);
00213
00214             if (s.soil < SOIL_THRESHOLD_LOW)
00215                 msg += String("ðŸš¨ Suelo muy seco, Bomba encendida: ") + s.soil +
"%\n";
00216             else if (s.soil > SOIL_THRESHOLD_HIGH)
```

```
00217                 msg += String("ðŸŒŠ Suelo demasiado hÃºmedo: ") + s.soil + "%\n";
00218
00219             if (msg.length()) {
00220                 char* buf = (char*)malloc(msg.length() + 1);
00221                 msg.toCharArray(buf, msg.length() + 1);
00222                 xQueueSend(msgQueue, &buf, portMAX_DELAY);
00223             }
00224         }
00225     }
00226 }
00227
00228 // ── Tarea para envío de mensajes por Telegram ──
00233 void TelegramTask(void *parameter) {
00234     char* recvBuffer;
00235     for (;;) {
00236         if (xQueueReceive(msgQueue, &recvBuffer, portMAX_DELAY)) {
00237             bool ok = bot.sendMessage(CHAT_ID, String(recvBuffer), "");
00238             Serial.println(ok ? "âœ… Telegram enviado" : "â□Œ Error en Telegram");
00239             free(recvBuffer);
00240         }
00241     }
00242 }
00243
00244 // ── Setup principal ──
00251 void setup() {
00252     Serial.begin(115200);
00253
00254     // Conexión Wi-Fi
00255     WiFi.mode(WIFI_STA);
00256     WiFi.begin(ssid, password);
00257     while (WiFi.status() != WL_CONNECTED) {
00258         delay(500);
00259     }
00260     client.setInsecure();
00261     bot.sendMessage(CHAT_ID, "ðŸ¤– Bot receptor iniciado", "");
00262
00263     // Sincronización de hora con NTP
00264     configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
00265     struct tm timeinfo;
00266     if (!getLocalTime(&timeinfo)) {
00267         Serial.println("â□Œ Error al obtener la hora del servidor NTP");
00268         bot.sendMessage(CHAT_ID, "â□Œ Error al obtener la hora desde el servidor
NTP", "");
00269     } else {
00270         Serial.printf("â□³ Hora obtenida: %02d:%02d:%02d\n", timeinfo.tm_hour,
timeinfo.tm_min, timeinfo.tm_sec);
00271         bot.sendMessage(CHAT_ID, "â□³ Hora NTP obtenida correctamente", "");
00272     }
00273
00274     // Inicio de SD_MMC
00275     if (!SD_MMC.begin()) {
00276         Serial.println("â□Œ Error al montar SD_MMC");
00277         bot.sendMessage(CHAT_ID, "â□Œ Error al montar la SD_MMC", "");
00278     } else {
00279         Serial.println("âœ… SD_MMC montada correctamente");
00280         bot.sendMessage(CHAT_ID, "âœ… SD_MMC montada correctamente", "");
00281     }
00282
00283     // Creación de colas y tareas
00284     dhtQueue  = xQueueCreate(5, sizeof(dht message t));
00285     soilQueue = xQueueCreate(5, sizeof(soil_message_t));
00286     msgQueue  = xQueueCreate(5, sizeof(char*));
00287     xTaskCreate(DHTTask, "DHTTask", 4096, NULL, 1, NULL);
00288     xTaskCreate(SoilTask, "SoilTask", 4096, NULL, 1, NULL);
00289     xTaskCreate(TelegramTask, "TelegramTask", 8192, NULL, 1, NULL);
00290
00291     // Configuración ESP-NOW
00292     if (esp_now_init() != ESP_OK) {
00293         Serial.println("â□Œ Error al iniciar ESP-NOW");
00294         return;
00295     }
00296     esp now register recv cb(OnDataRecv);
00297
00298     esp_now_peer_info_t peer = {};
00299     peer.channel = WiFi.channel();
00300     peer.encrypt = false;
00301     memcpy(peer.peer_addr, unifiedSenderMAC, 6);
```

```
00302     esp_now_add_peer(&peer);
00303     memcpy(peer.peer_addr, soilSenderMAC, 6);
00304     esp_now_add_peer(&peer);
00305 }
00306
00311 void loop() {
00312     vTaskDelay(portMAX_DELAY);
00313 }
```