IoT Environmental Station

1.0

Generated by Doxygen 1.14.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 dht_data_t Struct Reference

Structure to hold DHT sensor data with timestamp.

```
#include <dht_manager.h>
```

Data Fields

- float temperature
- float humidity
- char timestamp [ISO8601_STR_LEN]

3.1.1 Detailed Description

Structure to hold DHT sensor data with timestamp.

This structure contains temperature and humidity readings from a DHT sensor along with an ISO 8601 formatted timestamp indicating when the reading was taken.

Definition at line 17 of file dht_manager.h.

3.1.2 Field Documentation

3.1.2.1 humidity

```
float dht_data_t::humidity
```

Relative humidity reading in percentage (0-100%)

Definition at line 20 of file dht_manager.h.

3.1.2.2 temperature

```
float dht_data_t::temperature
```

Temperature reading in degrees Celsius

Definition at line 19 of file dht_manager.h.

3.1.2.3 timestamp

```
char dht_data_t::timestamp[ISO8601_STR_LEN]
```

ISO 8601 format timestamp (YYYY-MM-DDTHH:MM:SSZ)

Definition at line 21 of file dht_manager.h.

The documentation for this struct was generated from the following file:

- includes/dht_manager.h

Chapter 4

File Documentation

4.1 includes/common.c File Reference

```
#include "common.h"
```

Functions

- static void time_sync_notification_cb (struct timeval ∗tv)
- void setup_sntp (void)

  Initializes and configures the SNTP (Simple Network Time Protocol) client.
- void get_current_date_time (char ∗date_time)

  Retrieves the current date and time in ISO 8601 format.
- char ∗ create_json_payload (const dht_data_t ∗data)

  Creates a JSON payload string from DHT sensor data.

4.1.1 Function Documentation

4.1.1.1 create_json_payload()

```
char * create_json_payload (
            const dht_data_t * data)
```

Creates a JSON payload string from DHT sensor data.

This function takes DHT sensor data (temperature, humidity, timestamp) and creates a JSON formatted string suitable for transmission via MQTT or other communication protocols.

Parameters

	data	Pointer to a dht_data_t structure containing sensor readings and timestamp information

Returns

Pointer to a dynamically allocated string containing the JSON payload. The caller is responsible for freeing this memory. Returns NULL on error.

Definition at line 45 of file common.c.

4.1.1.2 get_current_date_time()

```
void get_current_date_time (
             char * date_time)
```

Retrieves the current date and time in ISO 8601 format.

This function gets the current system time and formats it as an ISO 8601 timestamp string (YYYY-MM-DDTHH↵:MM:SSZ format).

Parameters

	date_time	Pointer to a character buffer where the formatted date/time string will be stored. Buffer must be at least ISO8601_ST

Definition at line 36 of file common.c.

4.1.1.3 setup_sntp()

```
void setup_sntp (
             void )
```

Initializes and configures the SNTP (Simple Network Time Protocol) client.

This function sets up the SNTP client to synchronize the system time with internet time servers. It configures the timezone and starts the SNTP service.

Definition at line 9 of file common.c.

4.1.1.4 time_sync_notification_cb()

```
void time_sync_notification_cb (
             struct timeval * tv)  [static]
```

Definition at line 3 of file common.c.

4.2 common.c

Go to the documentation of this file.

```
00001 #include "common.h"
00002
00003 static void time_sync_notification_cb(struct timeval *tv)
00004 {
00005     const char *TAG = "SNTP Notification";
00006     ESP_LOGI(TAG, "Notification of a time synchronization event");
00007 }
00008
00009 void setup_sntp(void)
00010 {
00011     const char *TAG = "SNTP Initialization";
00012     ESP_LOGI(TAG, "Initializing SNTP");
00013     esp_sntp_setoperatingmode(SNTP_OPMODE_POLL);
00014     esp_sntp_setservername(0, "pool.ntp.org");
00015     sntp_set_time_sync_notification_cb(time_sync_notification_cb);
00016     esp_sntp_init();
00017
00018     // Set timezone to Spanish Peninsula Standard Time
00019     setenv("TZ", "CET-1CEST,M3.5.0/2,M10.5.0/3", 1);
```

```
    tzset();

    time_t now = 0;
    struct tm timeinfo = {0};
    int retry = 0;
    const int retry_count = 10;

    while (timeinfo.tm_year < (2020 - 1900) && ++retry < retry_count)
    {
        ESP_LOGI(TAG, "Waiting for system time to be set... (%d)", retry);
        vTaskDelay(pdMS_TO_TICKS(2000));
        time(&now);
        localtime_r(&now, &timeinfo);
    }
}

void get_current_date_time(char *date_time)
{
    time_t now;
    struct tm timeinfo;
    time(&now);
    localtime_r(&now, &timeinfo);
    strftime(date_time, ISO8601_STR_LEN, "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
}

char *create_json_payload(const dht_data_t *data)
{
    cJSON *root = cJSON_CreateObject();
    char tempStr[7], humStr[7];
    sprintf(tempStr, "%.2f", data->temperature);
    sprintf(humStr, "%.2f", data->humidity);

    cJSON_AddStringToObject(root, "temperature", tempStr);
    cJSON_AddStringToObject(root, "humidity", humStr);
    cJSON_AddStringToObject(root, "timestamp", data->timestamp);

    // Convert to string
    char *json_str = cJSON_PrintUnformatted(root);
    cJSON_Delete(root);

    return json_str;
}
```

4.3 includes/common.h File Reference

```
#include "dht_manager.h"
#include "esp_sntp.h"
#include "cJSON.h"
#include "esp_log.h"
```

Macros

- #define MEASURE_INTERVAL 60 * 1000
- #define WIFI_RECONNECT_INTERVAL_MS 60 * 1000
- #define TIMER_ID 1
- #define STACK_SIZE 4 * 1024
- #define MAX_Q_SIZE 10

Functions

- void setup_sntp (void)

  Initializes and configures the SNTP (Simple Network Time Protocol) client.
- void get_current_date_time (char *date_time)

  Retrieves the current date and time in ISO 8601 format.
- char * create_json_payload (const dht_data_t *data)

  Creates a JSON payload string from DHT sensor data.

4.3.1 Macro Definition Documentation

4.3.1.1 MAX_Q_SIZE

```
#define MAX_Q_SIZE 10
```

Definition at line 13 of file common.h.

4.3.1.2 MEASURE_INTERVAL

```
#define MEASURE_INTERVAL 60 * 1000
```

Definition at line 9 of file common.h.

4.3.1.3 STACK_SIZE

```
#define STACK_SIZE 4 * 1024
```

Definition at line 12 of file common.h.

4.3.1.4 TIMER_ID

```
#define TIMER_ID 1
```

Definition at line 11 of file common.h.

4.3.1.5 WIFI_RECONNECT_INTERVAL_MS

```
#define WIFI_RECONNECT_INTERVAL_MS 60 * 1000
```

Definition at line 10 of file common.h.

4.3.2 Function Documentation

4.3.2.1 create_json_payload()

```
char * create_json_payload (
             const dht_data_t * data)
```

Creates a JSON payload string from DHT sensor data.

This function takes DHT sensor data (temperature, humidity, timestamp) and creates a JSON formatted string suitable for transmission via MQTT or other communication protocols.

Parameters

	data	Pointer to a dht_data_t structure containing sensor readings and timestamp information

Returns

> Pointer to a dynamically allocated string containing the JSON payload. The caller is responsible for freeing this memory. Returns NULL on error.

Definition at line 45 of file common.c.

4.3.2.2 get_current_date_time()

```
void get_current_date_time (
            char * date_time)
```

Retrieves the current date and time in ISO 8601 format.

This function gets the current system time and formats it as an ISO 8601 timestamp string (YYYY-MM-DDTHH:MM:SSZ format).

Parameters

	date_time	Pointer to a character buffer where the formatted date/time string will be stored. Buffer must be at least ISO8601_ST

Definition at line 36 of file common.c.

4.3.2.3 setup_sntp()

```
void setup_sntp (
            void )
```

Initializes and configures the SNTP (Simple Network Time Protocol) client.

This function sets up the SNTP client to synchronize the system time with internet time servers. It configures the timezone and starts the SNTP service.

Definition at line 9 of file common.c.

4.4 common.h

Go to the documentation of this file.

```
#ifndef COMMON_H
#define COMMON_H

#include "dht_manager.h"
#include "esp_sntp.h"
#include "cJSON.h"
#include "esp_log.h"

#define MEASURE_INTERVAL 60 * 1000
#define WIFI_RECONNECT_INTERVAL_MS 60 * 1000
#define TIMER_ID 1
#define STACK_SIZE 4 * 1024
#define MAX_Q_SIZE 10

void setup_sntp(void);

void get_current_date_time(char *date_time);

char *create_json_payload(const dht_data_t *data);

#endif
```

4.5 includes/dht_manager.c File Reference

```
#include "dht_manager.h"
```

Functions

- esp_err_t setup_dht (void)

  Initializes the DHT sensor for temperature and humidity readings.

4.5.1 Function Documentation

4.5.1.1 setup_dht()

```
esp_err_t setup_dht (
             void )
```

Initializes the DHT sensor for temperature and humidity readings.

This function configures the DHT sensor (DHT11, DHT22, AM2301, or SI7021) based on the compile-time configuration. It sets up the GPIO pin and prepares the sensor for data acquisition.

Returns

ESP_OK on successful initialization, ESP_FAIL or other ESP error codes on failure

Definition at line 4 of file dht_manager.c.

4.6 dht_manager.c

Go to the documentation of this file.

```
00001
00002 #include "dht_manager.h"
00003
00004 esp_err_t setup_dht(void)
00005 {
00006     esp_err_t res;
00007     // Enable internal pull-up resistor if specified in menuconfig
00008     if (CONFIG_EXAMPLE_INTERNAL_PULLUP)
00009     {
00010         res = gpio_pullup_en(CONFIG_ESP_TEMP_SENSOR_GPIO);
00011     }
00012     else
00013     {
00014         res = gpio_pullup_dis(CONFIG_ESP_TEMP_SENSOR_GPIO);
00015     }
00016     return res;
00017 }
```

4.7 includes/dht_manager.h File Reference

```
#include "dht.h"
```

Data Structures

- struct dht_data_t

  Structure to hold DHT sensor data with timestamp.

Macros

- #define CONFIG_EXAMPLE_INTERNAL_PULLUP 0
- #define CONFIG_EXAMPLE_TYPE_DHT11 0
- #define ISO8601_STR_LEN 25
- #define DHT_SENSOR_TYPE (CONFIG_EXAMPLE_TYPE_AM2301 ? DHT_TYPE_AM2301 ↩ : CONFIG_EXAMPLE_TYPE_DHT11 ? DHT_TYPE_DHT11 : DHT_TYPE_SI7021)

Functions

- esp_err_t setup_dht (void)

  Initializes the DHT sensor for temperature and humidity readings.

4.7.1 Macro Definition Documentation

4.7.1.1 CONFIG_EXAMPLE_INTERNAL_PULLUP

```
#define CONFIG_EXAMPLE_INTERNAL_PULLUP 0
```

Definition at line 6 of file dht_manager.h.

4.7.1.2 CONFIG_EXAMPLE_TYPE_DHT11

```
#define CONFIG_EXAMPLE_TYPE_DHT11 0
```

Definition at line 7 of file dht_manager.h.

4.7.1.3 DHT_SENSOR_TYPE

```
#define DHT_SENSOR_TYPE (CONFIG_EXAMPLE_TYPE_AM2301 ? DHT_TYPE_AM2301 : CONFIG_EXAMPLE_TYPE_DHT11
? DHT_TYPE_DHT11 : DHT_TYPE_SI7021)
```

Definition at line 9 of file dht_manager.h.

4.7.1.4 ISO8601_STR_LEN

```
#define ISO8601_STR_LEN 25
```

Definition at line 8 of file dht_manager.h.

4.7.2 Function Documentation

4.7.2.1 setup_dht()

```
esp_err_t setup_dht (
            void )
```

Initializes the DHT sensor for temperature and humidity readings.

This function configures the DHT sensor (DHT11, DHT22, AM2301, or SI7021) based on the compile-time configuration. It sets up the GPIO pin and prepares the sensor for data acquisition.

Returns

ESP_OK on successful initialization, ESP_FAIL or other ESP error codes on failure

Definition at line 4 of file dht_manager.c.

4.8 dht_manager.h

Go to the documentation of this file.

```
#ifndef DHT_SENSOR_H
#define DHT_SENSOR_H

#include "dht.h"

#define CONFIG_EXAMPLE_INTERNAL_PULLUP 0
#define CONFIG_EXAMPLE_TYPE_DHT11 0
#define ISO8601_STR_LEN 25 // "YYYY-MM-DDTHH:MM:SSZ" + null
#define DHT_SENSOR_TYPE (CONFIG_EXAMPLE_TYPE_AM2301 ? DHT_TYPE_AM2301 : CONFIG_EXAMPLE_TYPE_DHT11 ?
DHT_TYPE_DHT11 : DHT_TYPE_SI7021)

typedef struct
{
    float temperature;
    float humidity;
    char timestamp[ISO8601_STR_LEN];
} dht_data_t;

esp_err_t setup_dht(void);

#endif
```

4.9 includes/display_manager.c File Reference

```
#include "display_manager.h"
```

Functions

- u8g2_t init_oled_display (void)

  Initializes the OLED display using the u8g2 graphics library.
- void show_dht_data (u8g2_t u8g2, const char ∗date, const char ∗time, const char ∗temp, const char ∗hum)

  Displays DHT sensor data on the OLED screen.

4.9.1 Function Documentation

4.9.1.1 init_oled_display()

```
u8g2_t init_oled_display (
            void )
```

Initializes the OLED display using the u8g2 graphics library.

This function sets up the I2C communication interface and initializes the OLED display using the u8g2 library. It configures the display for subsequent drawing operations.

Returns

u8g2_t structure containing the display configuration and state. This structure is used for all subsequent display operations.

Definition at line 3 of file display_manager.c.

4.9.1.2 show_dht_data()

```
void show_dht_data (
            u8g2_t u8g2,
            const char * date,
            const char * time,
            const char * temp,
            const char * hum)
```

Displays DHT sensor data on the OLED screen.

This function renders the date, time, temperature, and humidity values on the OLED display using the u8g2 graphics library. It formats and positions the text appropriately on the screen.

Parameters

	u8g2	The u8g2 display structure initialized by init_oled_display()
	date	Pointer to a null-terminated string containing the date information
	time	Pointer to a null-terminated string containing the time information
	temp	Pointer to a null-terminated string containing the temperature reading
	hum	Pointer to a null-terminated string containing the humidity reading

Definition at line 34 of file display_manager.c.

4.10 display_manager.c

Go to the documentation of this file.

```
#include "display_manager.h"

u8g2_t init_oled_display(void)
{
    static const char *TAG = "oled_display_module";
    /* OLED Display Setup*/
    u8g2_t u8g2;
    u8g2_esp32_hal_t u8g2_esp32_hal = U8G2_ESP32_HAL_DEFAULT;
    u8g2_esp32_hal.bus.i2c.scl = I2C_SCL;
    u8g2_esp32_hal.bus.i2c.sda = I2C_SDA;
    u8g2_esp32_hal_init(u8g2_esp32_hal);

    u8g2_Setup_ssd1306_i2c_128x64_noname_f(&u8g2, U8G2_R0,
                                           // u8x8_byte_sw_i2c,
                                           u8g2_esp32_i2c_byte_cb,
                                           u8g2_esp32_gpio_and_delay_cb);
    u8x8_SetI2CAddress(&u8g2.u8x8, 0x78);
    ESP_LOGI(TAG, "Init Display");
    u8g2_InitDisplay(&u8g2); // send init sequence to the display, display is in
                             // sleep mode after this,

    ESP_LOGI(TAG, "WakeUp Display");
    u8g2_SetPowerSave(&u8g2, 0); // wake up display
    ESP_LOGI(TAG, "Clear Buffer");

    u8g2_SetFont(&u8g2, u8g2_font_unifont_t_symbols);

    u8g2_ClearBuffer(&u8g2);
    /* End of setup*/

    return u8g2;
}

void show_dht_data(u8g2_t u8g2, const char *date, const char *time, const char *temp, const char *hum)
{
    u8g2_ClearBuffer(&u8g2);
    u8g2_DrawStr(&u8g2, 0, 12, date);
    u8g2_DrawStr(&u8g2, 0, 28, time);
    u8g2_DrawStr(&u8g2, 0, 46, temp);
    u8g2_DrawGlyph(&u8g2, 110, 46, 0x2600);
    u8g2_DrawStr(&u8g2, 0, 62, hum);
    u8g2_DrawGlyph(&u8g2, 110, 62, 0x2614);
    u8g2_SendBuffer(&u8g2); // Send the buffer data to display
}
```

4.11 includes/display_manager.h File Reference

```
#include "driver/i2c_master.h"
#include "u8g2.h"
#include "u8g2_esp32_hal.h"
#include "esp_log.h"
```

Macros

- #define I2C_SDA 8
- #define I2C_SCL 9

Functions

- u8g2_t init_oled_display (void)

    Initializes the OLED display using the u8g2 graphics library.
- void show_dht_data (u8g2_t u8g2, const char ∗date, const char ∗time, const char ∗temp, const char ∗hum)

    Displays DHT sensor data on the OLED screen.

4.11.1 Macro Definition Documentation

4.11.1.1 I2C_SCL

```
#define I2C_SCL 9
```

I2C SCL (Serial Clock) pin number

Definition at line 10 of file display_manager.h.

4.11.1.2 I2C_SDA

```
#define I2C_SDA 8
```

I2C SDA (Serial Data) pin number

Definition at line 9 of file display_manager.h.

4.11.2 Function Documentation

4.11.2.1 init_oled_display()

```
u8g2_t init_oled_display (
            void )
```

Initializes the OLED display using the u8g2 graphics library.

This function sets up the I2C communication interface and initializes the OLED display using the u8g2 library. It configures the display for subsequent drawing operations.

Returns

u8g2_t structure containing the display configuration and state. This structure is used for all subsequent display operations.

Definition at line 3 of file display_manager.c.

4.11.2.2 show_dht_data()

```
void show_dht_data (
            u8g2_t u8g2,
            const char * date,
            const char * time,
            const char * temp,
            const char * hum)
```

Displays DHT sensor data on the OLED screen.

This function renders the date, time, temperature, and humidity values on the OLED display using the u8g2 graphics library. It formats and positions the text appropriately on the screen.

Parameters

u8g2	The u8g2 display structure initialized by init_oled_display()
date	Pointer to a null-terminated string containing the date information
time	Pointer to a null-terminated string containing the time information
temp	Pointer to a null-terminated string containing the temperature reading
hum	Pointer to a null-terminated string containing the humidity reading

Definition at line 34 of file display_manager.c.

4.12 display_manager.h

Go to the documentation of this file.

```
#ifndef DISPLAY_H
#define DISPLAY_H

#include "driver/i2c_master.h"
#include "u8g2.h"
#include "u8g2_esp32_hal.h"
#include "esp_log.h"

#define I2C_SDA 8
#define I2C_SCL 9

u8g2_t init_oled_display(void);

void show_dht_data(u8g2_t u8g2, const char *date, const char *time, const char *temp, const char *hum);

#endif
```

4.13 includes/mqtt_manager.c File Reference

```
#include "mqtt_manager.h"
```

Functions

- static void mqtt_event_handler (void ∗handler_args, esp_event_base_t base, int32_t event_id, void ∗event_data)
- void setup_mqtt (void)

  Initializes and configures the MQTT client for IoT communication.

Variables

- bool MQTT_CONNECTED = false
- esp_mqtt_client_handle_t client = NULL

4.13.1 Function Documentation

4.13.1.1 mqtt_event_handler()

```
void mqtt_event_handler (
             void * handler_args,
             esp_event_base_t base,
             int32_t event_id,
             void * event_data)  [static]
```

Definition at line 7 of file mqtt_manager.c.

4.13.1.2 setup_mqtt()

```
void setup_mqtt (
             void )
```

Initializes and configures the MQTT client for IoT communication.

This function sets up the MQTT client with the necessary configuration including broker connection details and event handlers. It establishes the connection to the MQTT broker and prepares the client for publishing sensor data and receiving commands. The configuration parameters are typically read from the ESP-IDF configuration system (menuconfig).

Definition at line 49 of file mqtt_manager.c.

4.13.2 Variable Documentation

4.13.2.1 client

```
esp_mqtt_client_handle_t client = NULL
```

MQTT client handle

Definition at line 5 of file mqtt_manager.c.

4.13.2.2 MQTT_CONNECTED

```
bool MQTT_CONNECTED = false
```

MQTT connection status flag

Definition at line 4 of file mqtt_manager.c.

4.14 mqtt_manager.c

Go to the documentation of this file.

```
#include "mqtt_manager.h"


bool MQTT_CONNECTED = false;
esp_mqtt_client_handle_t client = NULL;

static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void *event_data)
{
    const char *TAG = "MQTT_EVENT_HANDLER";
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    int msg_id;
    switch ((esp_mqtt_event_id_t)event_id)
    {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
        MQTT_CONNECTED = true;

        msg_id = esp_mqtt_client_subscribe(client, "/home/office/dht", 0);
        ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGW(TAG, "MQTT_EVENT_DISCONNECTED");
        MQTT_CONNECTED = false;
        break;

    case MQTT_EVENT_SUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGW(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG, "TOPIC=%.*s", event->topic_len, event->topic);
        ESP_LOGI(TAG, "DATA=%.*s", event->data_len, event->data);
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
        break;
    default:
        ESP_LOGI(TAG, "Other event id:%d", event->event_id);
        break;
    }
}

void setup_mqtt(void)
{
    const char *TAG = "Setup MQTT";
    ESP_LOGI(TAG, "STARTING MQTT");
    esp_mqtt_client_config_t mqttConfig = {
        .broker.address.uri = CONFIG_BROKER_URI,
    };

    client = esp_mqtt_client_init(&mqttConfig);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
    esp_mqtt_client_start(client);
}
```

4.15 includes/mqtt_manager.h File Reference

```
#include "mqtt_client.h"
#include "esp_log.h"
```

Functions

- void setup_mqtt (void)

  Initializes and configures the MQTT client for IoT communication.

4.15.1 Function Documentation

4.15.1.1 setup_mqtt()

```
void setup_mqtt (
            void )
```

Initializes and configures the MQTT client for IoT communication.

This function sets up the MQTT client with the necessary configuration including broker connection details and event handlers. It establishes the connection to the MQTT broker and prepares the client for publishing sensor data and receiving commands. The configuration parameters are typically read from the ESP-IDF configuration system (menuconfig).

Definition at line 49 of file mqtt_manager.c.

4.16 mqtt_manager.h

Go to the documentation of this file.

```
#ifndef MQTT_MANAGER_H
#define MQTT_MANAGER_H

#include "mqtt_client.h"
#include "esp_log.h"

void setup_mqtt(void);

#endif
```

4.17 includes/wifi_manager.c File Reference

```
#include "wifi_manager.h"
```

Functions

- static void wifi_event_handler (void ∗arg, esp_event_base_t event_base, int32_t event_id, void ∗event_data)
- void setup_wifi (void)

    Initializes and configures WiFi connectivity for the ESP32.

Variables

- EventGroupHandle_t s_wifi_event_group

4.17.1 Function Documentation

4.17.1.1 setup_wifi()

```
void setup_wifi (
            void )
```

Initializes and configures WiFi connectivity for the ESP32.

This function sets up the WiFi subsystem including:

- WiFi driver initialization
- Event loop configuration
- Access Point connection with credentials from configuration
- Retry mechanism for failed connections
- Event handling for connection status

The function uses configuration parameters (SSID, password, security mode) defined through the ESP-IDF configuration system. It will attempt to connect to the configured access point and handle reconnection attempts according to the maximum retry settings.

Definition at line 28 of file wifi_manager.c.

4.17.1.2 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data) [static]
```

Definition at line 6 of file wifi_manager.c.

4.17.2 Variable Documentation

4.17.2.1 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group
```

WiFi event group handle

Definition at line 4 of file wifi_manager.c.

4.18 wifi_manager.c

Go to the documentation of this file.

```c
#include "wifi_manager.h"

/* FreeRTOS event group to signal when we are connected*/
EventGroupHandle_t s_wifi_event_group;

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{   const char *TAG = "Wifi Event Handler";
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
        esp_wifi_connect();
    }
    else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
    {
        ESP_LOGI(TAG, "Disconnected from AP, will try to reconnect in 60 seconds");
        xEventGroupClearBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
    else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        // initialize_sntp();
        // mqtt_app_start();
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

void setup_wifi(void)
{
    const char *TAG = "Setup Wifi";
    ESP_LOGI(TAG, "Setting up WiFi");
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &wifi_event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
            /* Authmode threshold resets to WPA2 as default if password matches WPA2 standards (password len => 8).
             * If you want to connect the device to deprecated WEP/WPA networks, Please set the threshold value
             * to WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK and set the password with length and format matching to
             * WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK standards.
             */
            .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
            .sae_pwe_h2e = ESP_WIFI_SAE_MODE,
            .sae_h2e_identifier = EXAMPLE_H2E_IDENTIFIER,
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "wifi_task started and wifi driver started");
}
```

4.19 includes/wifi_manager.h File Reference

```
#include "esp_log.h"
#include "esp_wifi.h"
#include "freertos/event_groups.h"
#include "freertos/FreeRTOS.h"
```

Macros

- #define EXAMPLE_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
- #define EXAMPLE_ESP_WIFI_PASS CONFIG_ESP_WIFI_PASSWORD
- #define EXAMPLE_ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY
- #define WIFI_CONNECTED_BIT BIT0

Functions

- void setup_wifi (void)

  Initializes and configures WiFi connectivity for the ESP32.

4.19.1 Macro Definition Documentation

4.19.1.1 EXAMPLE_ESP_MAXIMUM_RETRY

```
#define EXAMPLE_ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY
```

Definition at line 11 of file wifi_manager.h.

4.19.1.2 EXAMPLE_ESP_WIFI_PASS

```
#define EXAMPLE_ESP_WIFI_PASS CONFIG_ESP_WIFI_PASSWORD
```

Definition at line 10 of file wifi_manager.h.

4.19.1.3 EXAMPLE_ESP_WIFI_SSID

```
#define EXAMPLE_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
```

Definition at line 9 of file wifi_manager.h.

4.19.1.4 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```

Event bit indicating successful WiFi connection

Definition at line 44 of file wifi_manager.h.

4.19.2 Function Documentation

4.19.2.1 setup_wifi()

```
void setup_wifi (
             void )
```

Initializes and configures WiFi connectivity for the ESP32.

This function sets up the WiFi subsystem including:

- WiFi driver initialization
- Event loop configuration
- Access Point connection with credentials from configuration
- Retry mechanism for failed connections
- Event handling for connection status

The function uses configuration parameters (SSID, password, security mode) defined through the ESP-IDF configuration system. It will attempt to connect to the configured access point and handle reconnection attempts according to the maximum retry settings.

Definition at line 28 of file wifi_manager.c.

4.20 wifi_manager.h

Go to the documentation of this file.

```
#ifndef WIFI_MANAGER_H
#define WIFI_MANAGER_H

#include "esp_log.h"
#include "esp_wifi.h"
#include "freertos/event_groups.h"
#include "freertos/FreeRTOS.h"

#define EXAMPLE_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
#define EXAMPLE_ESP_WIFI_PASS CONFIG_ESP_WIFI_PASSWORD
#define EXAMPLE_ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY

#if CONFIG_ESP_WPA3_SAE_PWE_HUNT_AND_PECK
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HUNT_AND_PECK
#define EXAMPLE_H2E_IDENTIFIER ""
#elif CONFIG_ESP_WPA3_SAE_PWE_HASH_TO_ELEMENT
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HASH_TO_ELEMENT
#define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
#elif CONFIG_ESP_WPA3_SAE_PWE_BOTH
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_BOTH
#define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
#endif
#if CONFIG_ESP_WIFI_AUTH_OPEN
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
#elif CONFIG_ESP_WIFI_AUTH_WEP
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
#elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
```

```
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
#endif

/* The event group allows multiple bits for each event, but we only care about two events:
 * - we are connected to the AP with an IP
 * - we failed to connect after the maximum amount of retries */
#define WIFI_CONNECTED_BIT BIT0

void setup_wifi(void);

#endif
```

4.21 main/main.c File Reference

IoT Environmental Station Main Application.

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "dht_manager.h"
#include "display_manager.h"
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "common.h"
#include "freertos/task.h"
#include "freertos/timers.h"
#include "freertos/queue.h"
#include "esp_system.h"
#include "esp_event.h"
#include "nvs_flash.h"
```

Functions

- esp_err_t setup_timer (void)

  Creates and starts the DHT sensor measurement timer.
- esp_err_t create_tasks (void)

  Creates and starts all FreeRTOS tasks for the application.
- void measure_temp_hum (TimerHandle_t timer)

  Timer callback function for reading DHT sensor data.
- void task_show_data_oled (void ∗args)

  FreeRTOS task for displaying sensor data on OLED screen.
- void task_send_data_mqtt (void ∗args)

  FreeRTOS task for transmitting sensor data via MQTT.
- void task_wifi (void ∗args)

  FreeRTOS task for WiFi connection management.
- void app_main (void)

  Main application entry point.

Variables

- EventGroupHandle_t s_wifi_event_group
- bool MQTT_CONNECTED
- esp_mqtt_client_handle_t client
- TimerHandle_t timerDHT
- QueueHandle_t displayQueue
- QueueHandle_t mqttQueue
- static const char ∗ TAG = "iot_env_station"

4.21.1 Detailed Description

IoT Environmental Station Main Application.

This file contains the main application logic for an IoT environmental monitoring station that reads temperature and humidity data from a DHT sensor, displays the information on an OLED screen, and transmits the data via MQTT over WiFi.

The application uses FreeRTOS tasks for concurrent operation:

- Timer-based sensor reading
- OLED display updates
- MQTT data transmission
- WiFi connection management

Definition in file main.c.

4.21.2 Function Documentation

4.21.2.1 app_main()

```
void app_main (
             void )
```

Main application entry point.

This function initializes the IoT environmental station by:

- Setting up NVS (Non-Volatile Storage) for configuration data
- Creating FreeRTOS queues for inter-task communication
- Initializing the DHT sensor
- Setting up the measurement timer
- Creating and starting all application tasks

The function handles NVS initialization errors by erasing and reinitializing the flash if necessary.

Definition at line 67 of file main.c.

4.21.2.2 create_tasks()

```
esp_err_t create_tasks (
            void )
```

Creates and starts all FreeRTOS tasks for the application.

This function creates three main tasks:

1. Display task (priority 1): Updates OLED display with sensor data
2. WiFi task (priority 3): Manages WiFi connection and reconnection
3. MQTT task (priority 2): Handles MQTT communication and data transmission

Each task is created with a stack size defined by STACK_SIZE and assigned appropriate priorities for proper system operation.

Returns

ESP_OK if all tasks are created successfully, ESP_FAIL if any task creation fails

Definition at line 133 of file main.c.

4.21.2.3 measure_temp_hum()

```
void measure_temp_hum (
            TimerHandle_t timer)
```

Timer callback function for reading DHT sensor data.

This function is called periodically by the FreeRTOS timer to:

- Read temperature and humidity data from the DHT sensor
- Capture the current timestamp in ISO8601 format
- Package the data into a dht_data_t structure
- Send the data to both display and MQTT queues for processing

The function handles read errors by logging appropriate error messages and only sends data to queues when the sensor reading is successful. Queue send operations use a timeout to prevent blocking if queues are full.

Parameters

	timer	Handle to the timer that triggered this callback (unused)

Definition at line 318 of file main.c.

4.21.2.4 setup_timer()

```
esp_err_t setup_timer (
            void )
```

Creates and starts the DHT sensor measurement timer.

This function creates a FreeRTOS software timer that triggers periodic temperature and humidity measurements from the DHT sensor. The timer is configured to repeat at intervals defined by MEASURE_INTERVAL and calls the measure_temp_hum() callback function.

Returns

ESP_OK on successful timer creation and start, ESP_FAIL on error

Definition at line 96 of file main.c.

4.21.2.5 task_send_data_mqtt()

```
void task_send_data_mqtt (
            void * args)
```

FreeRTOS task for transmitting sensor data via MQTT.

This task handles MQTT communication by:

- Waiting for WiFi connection establishment
- Initializing MQTT client and SNTP time synchronization
- Continuously monitoring the MQTT queue for sensor data
- Converting sensor data to JSON format
- Publishing data to the configured MQTT topic

The task only attempts to send data when both the queue has data and the MQTT client is connected to the broker.

Parameters

	args	Pointer to task parameters (unused in this implementation)

Definition at line 234 of file main.c.

4.21.2.6 task_show_data_oled()

```
void task_show_data_oled (
            void * args)
```

FreeRTOS task for displaying sensor data on OLED screen.

This task continuously monitors the display queue for new sensor data and updates the OLED display with formatted temperature, humidity, date, and time information. The task:

- Waits for data from the display queue
- Parses the ISO8601 timestamp from sensor data
- Formats date and time strings for display
- Updates the OLED screen with current readings

Parameters

	args	Pointer to task parameters (unused in this implementation)

Definition at line 187 of file main.c.

4.21.2.7 task_wifi()

```
void task_wifi (
             void * args)
```

FreeRTOS task for WiFi connection management.

This task manages the WiFi connection by:

- Initializing the WiFi subsystem and connecting to the configured network
- Continuously monitoring the connection status using event groups
- Automatically attempting reconnection if the connection is lost
- Implementing a watchdog mechanism with configurable reconnect intervals

The task uses the WIFI_RECONNECT_INTERVAL_MS timeout to check connection status and triggers reconnection attempts when necessary.

Parameters

	args	Pointer to task parameters (unused in this implementation)

Definition at line 282 of file main.c.

4.21.3 Variable Documentation

4.21.3.1 client

```
esp_mqtt_client_handle_t client  [extern]
```

MQTT client handle

Definition at line 5 of file mqtt_manager.c.

4.21.3.2 displayQueue

```
QueueHandle_t displayQueue
```

Queue for sensor data to be displayed on OLED

Definition at line 39 of file main.c.

4.21.3.3 MQTT_CONNECTED

```
bool MQTT_CONNECTED  [extern]
```

MQTT connection status flag

Definition at line 4 of file mqtt_manager.c.

4.21.3.4 mqttQueue

```
QueueHandle_t mqttQueue
```

Queue for sensor data to be sent via MQTT

Definition at line 40 of file main.c.

4.21.3.5 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [extern]
```

WiFi event group handle

Definition at line 4 of file wifi_manager.c.

4.21.3.6 TAG

```
const char* TAG = "iot_env_station"  [static]
```

Log tag for this module

Definition at line 42 of file main.c.

4.21.3.7 timerDHT

```
TimerHandle_t timerDHT
```

Timer handle for periodic DHT sensor readings

Definition at line 38 of file main.c.

4.22 main.c

Go to the documentation of this file.

```


#include <stdio.h>
#include <string.h>
#include <time.h>

#include "dht_manager.h"
#include "display_manager.h"
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "common.h"

#include "freertos/task.h"
#include "freertos/timers.h"
#include "freertos/queue.h"

#include "esp_system.h"
#include "esp_event.h"
#include "nvs_flash.h"

extern EventGroupHandle_t s_wifi_event_group;
extern bool MQTT_CONNECTED;
extern esp_mqtt_client_handle_t client;

TimerHandle_t timerDHT;
QueueHandle_t displayQueue;
QueueHandle_t mqttQueue;

static const char *TAG = "iot_env_station";

/* Function prototypes */
esp_err_t setup_timer(void);
esp_err_t create_tasks(void);

void measure_temp_hum(TimerHandle_t timer);
void task_show_data_oled(void *args);
void task_send_data_mqtt(void *args);
void task_wifi(void *args);

void app_main(void)
{
    // Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    displayQueue = xQueueCreate(MAX_Q_SIZE, sizeof(dht_data_t));
    mqttQueue = xQueueCreate(MAX_Q_SIZE, sizeof(dht_data_t));
    ESP_ERROR_CHECK(setup_dht());
    ESP_ERROR_CHECK(setup_timer());
    ESP_ERROR_CHECK(create_tasks());
}

esp_err_t setup_timer(void)
{
    timerDHT = xTimerCreate("Timer DHT",
                            pdMS_TO_TICKS(MEASURE_INTERVAL),
                            pdTRUE,
                            (void *)TIMER_ID,
                            measure_temp_hum);
    if (timerDHT == NULL)
    {
        ESP_LOGE(TAG, "Timer could not be created");
        return ESP_FAIL;
    }
    else
    {
        if (xTimerStart(timerDHT, 0) != pdPASS)
        {
            ESP_LOGE(TAG, "The timer could not be set into the Active state");
            return ESP_FAIL;
        }
    }
    return ESP_OK;
}

esp_err_t create_tasks(void)
{
    static uint8_t ucParameterToPass;
    TaskHandle_t displayHandle = NULL;
    TaskHandle_t mqttHandle = NULL;
    TaskHandle_t wifiHandle = NULL;

    if (xTaskCreate(task_show_data_oled,
                    "Show read data on oled display",
                    STACK_SIZE,
                    &ucParameterToPass,
                    1,
                    &displayHandle) != pdPASS)
    {
        return ESP_FAIL;
    }

    if (xTaskCreate(task_wifi,
                    "Task to connect to wifi",
                    STACK_SIZE,
                    &ucParameterToPass,
                    3,
                    &wifiHandle) != pdPASS)
    {
        return ESP_FAIL;
    }

    if (xTaskCreate(task_send_data_mqtt,
                    "Send data to MQTT broker",
                    STACK_SIZE,
                    &ucParameterToPass,
                    2,
                    &mqttHandle) != pdPASS)
    {
        return ESP_FAIL;
    }

    return ESP_OK;
}

void task_show_data_oled(void *args)
{
    dht_data_t sensorData = {0};
    u8g2_t u8g2;
    // OLED Display Setup
    u8g2 = init_oled_display();

    while (true)
    {
        if (xQueueReceive(displayQueue, &sensorData, portMAX_DELAY))
        {
            struct tm timeinfo;
            char temp_str[20], hum_str[20], date_str[17], time_str[12];
            // Parse ISO8601 string back to time
            strptime(sensorData.timestamp, "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
            // Format the date as "dd/mm/YYYY"
            strftime(date_str, sizeof(date_str), "Date: %d/%m/%Y", &timeinfo);
            // Format the time as "HH:mm"
            strftime(time_str, sizeof(time_str), "Time: %H:%M", &timeinfo);
            sprintf(temp_str, "Temp: %.2fC", sensorData.temperature);
            sprintf(hum_str, "Hum:  %.2f %%", sensorData.humidity);
            show_dht_data(u8g2, date_str, time_str, temp_str, hum_str);
        }
        else
        {
            ESP_LOGE(TAG, "Error receiving data or no data in buffer");
        }
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

void task_send_data_mqtt(void *args)
{
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT,
                                           pdFALSE,
                                           pdTRUE,
                                           pdMS_TO_TICKS(portMAX_DELAY));

    if ((bits & WIFI_CONNECTED_BIT) == 1)
    {
        ESP_LOGI(TAG, "WiFi connected, starting MQTT task and SNTP");
        setup_mqtt();
        setup_sntp();
    }

    dht_data_t sensorData = {0};
    while (true)
    {
        if ((xQueueReceive(mqttQueue, &sensorData, portMAX_DELAY)) && MQTT_CONNECTED)
        {
            // Convert to JSON string
            char *json_str = create_json_payload(&sensorData);
            esp_mqtt_client_publish(client, "/home/office/dht", json_str, 0, 0, 0);
            free(json_str);
        }
        else
        {
            ESP_LOGE(TAG, "Error receiving data or no data in buffer");
        }
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

void task_wifi(void *args)
{
    setup_wifi();

    while (true)
    {
        EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                               WIFI_CONNECTED_BIT,
                                               pdFALSE,
                                               pdFALSE,
                                               pdMS_TO_TICKS(WIFI_RECONNECT_INTERVAL_MS));

        if ((bits & WIFI_CONNECTED_BIT) == 0)
        {
            esp_wifi_connect();
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void measure_temp_hum(TimerHandle_t timer)
{
    esp_err_t res;
    dht_data_t dhtData;
    res = dht_read_float_data(DHT_SENSOR_TYPE, CONFIG_ESP_TEMP_SENSOR_GPIO, &dhtData.humidity, &dhtData.temperature);
    get_current_date_time(dhtData.timestamp);
    if (res == ESP_OK)
    {
        if (xQueueSend(displayQueue, &dhtData, pdMS_TO_TICKS(100)) != pdPASS)
        {
            ESP_LOGE(TAG, "Error sending data to display queue");
        }
        if (xQueueSend(mqttQueue, &dhtData, pdMS_TO_TICKS(100)) != pdPASS)
        {
            ESP_LOGE(TAG, "Error sending data to MQTT queue");
        }
    }
    else
    {
        ESP_LOGE(TAG, "Error reading data");
    }
}
```

Index