

IoT Environmental Station

1.0

Generated by Doxygen 1.14.0

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 dht_data_t Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 humidity	5
3.1.2.2 temperature	6
3.1.2.3 timestamp	6
4 File Documentation	7
4.1 includes/common.c File Reference	7
4.1.1 Function Documentation	7
4.1.1.1 create_json_payload()	7
4.1.1.2 get_current_date_time()	8
4.1.1.3 setup_sntp()	8
4.1.1.4 time_sync_notification_cb()	8
4.2 common.c	8
4.3 includes/common.h File Reference	9
4.3.1 Macro Definition Documentation	10
4.3.1.1 MAX_Q_SIZE	10
4.3.1.2 MEASURE_INTERVAL	10
4.3.1.3 STACK_SIZE	10
4.3.1.4 TIMER_ID	10
4.3.1.5 WIFI_RECONNECT_INTERVAL_MS	10
4.3.2 Function Documentation	10
4.3.2.1 create_json_payload()	10
4.3.2.2 get_current_date_time()	11
4.3.2.3 setup_sntp()	11
4.4 common.h	11
4.5 includes/dht_manager.c File Reference	11
4.5.1 Function Documentation	12
4.5.1.1 setup_dht()	12
4.6 dht_manager.c	12
4.7 includes/dht_manager.h File Reference	12
4.7.1 Macro Definition Documentation	13
4.7.1.1 CONFIG_EXAMPLE_INTERNAL_PULLUP	13
4.7.1.2 CONFIG_EXAMPLE_TYPE_DHT11	13
4.7.1.3 DHT_SENSOR_TYPE	13

4.7.1.4 ISO8601_STR_LEN	13
4.7.2 Function Documentation	14
4.7.2.1 setup_dht()	14
4.8 dht_manager.h	14
4.9 includes/display_manager.c File Reference	14
4.9.1 Function Documentation	15
4.9.1.1 init_oled_display()	15
4.9.1.2 show_dht_data()	15
4.10 display_manager.c	16
4.11 includes/display_manager.h File Reference	16
4.11.1 Macro Definition Documentation	17
4.11.1.1 I2C_SCL	17
4.11.1.2 I2C_SDA	17
4.11.2 Function Documentation	17
4.11.2.1 init_oled_display()	17
4.11.2.2 show_dht_data()	17
4.12 display_manager.h	18
4.13 includes/mqtt_manager.c File Reference	18
4.13.1 Function Documentation	19
4.13.1.1 mqtt_event_handler()	19
4.13.1.2 setup_mqtt()	19
4.13.2 Variable Documentation	19
4.13.2.1 client	19
4.13.2.2 MQTT_CONNECTED	19
4.14 mqtt_manager.c	20
4.15 includes/mqtt_manager.h File Reference	20
4.15.1 Function Documentation	21
4.15.1.1 setup_mqtt()	21
4.16 mqtt_manager.h	21
4.17 includes/wifi_manager.c File Reference	21
4.17.1 Function Documentation	22
4.17.1.1 setup_wifi()	22
4.17.1.2 wifi_event_handler()	22
4.17.2 Variable Documentation	22
4.17.2.1 s_wifi_event_group	22
4.18 wifi_manager.c	23
4.19 includes/wifi_manager.h File Reference	24
4.19.1 Macro Definition Documentation	24
4.19.1.1 EXAMPLE_ESP_MAXIMUM_RETRY	24
4.19.1.2 EXAMPLE_ESP_WIFI_PASS	24
4.19.1.3 EXAMPLE_ESP_WIFI_SSID	24
4.19.1.4 WIFI_CONNECTED_BIT	24

4.19.2 Function Documentation	25
4.19.2.1 setup_wifi()	25
4.20 wifi_manager.h	25
4.21 main/main.c File Reference	26
4.21.1 Detailed Description	27
4.21.2 Function Documentation	27
4.21.2.1 app_main()	27
4.21.2.2 create_tasks()	28
4.21.2.3 measure_temp_hum()	28
4.21.2.4 setup_timer()	29
4.21.2.5 task_send_data_mqtt()	29
4.21.2.6 task_show_data_oled()	29
4.21.2.7 task_wifi()	30
4.21.3 Variable Documentation	30
4.21.3.1 client	30
4.21.3.2 displayQueue	30
4.21.3.3 MQTT_CONNECTED	31
4.21.3.4 mqttQueue	31
4.21.3.5 s_wifi_event_group	31
4.21.3.6 TAG	31
4.21.3.7 timerDHT	31
4.22 main.c	32
Index	35

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

dht_data_t	Structure to hold DHT sensor data with timestamp	5
----------------------------	--	---

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

includes/ common.c	7
includes/ common.h	9
includes/ dht_manager.c	11
includes/ dht_manager.h	12
includes/ display_manager.c	14
includes/ display_manager.h	16
includes/ mqtt_manager.c	18
includes/ mqtt_manager.h	20
includes/ wifi_manager.c	21
includes/ wifi_manager.h	24
main/ main.c	
IoT Environmental Station Main Application	26

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

<code>date_time</code>	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);
```

```

00020     tzset();
00021
00022     time_t now = 0;
00023     struct tm timeinfo = {0};
00024     int retry = 0;
00025     const int retry_count = 10;
00026
00027     while (timeinfo.tm_year < (2020 - 1900) && ++retry < retry_count)
00028     {
00029         ESP_LOGI(TAG, "Waiting for system time to be set... (%d)", retry);
00030         vTaskDelay(pdMS_TO_TICKS(2000));
00031         time(&now);
00032         localtime_r(&now, &timeinfo);
00033     }
00034 }
00035
00036 void get_current_date_time(char *date_time)
00037 {
00038     time_t now;
00039     struct tm timeinfo;
00040     time(&now);
00041     localtime_r(&now, &timeinfo);
00042     strftime(date_time, ISO8601_STR_LEN, "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
00043 }
00044
00045 char *create_json_payload(const dht_data_t *data)
00046 {
00047     cJSON *root = cJSON_CreateObject();
00048     char tempStr[7], humStr[7];
00049     sprintf(tempStr, "%.2f", data->temperature);
00050     sprintf(humStr, "%.2f", data->humidity);
00051
00052     cJSON_AddStringToObject(root, "temperature", tempStr);
00053     cJSON_AddStringToObject(root, "humidity", humStr);
00054     cJSON_AddStringToObject(root, "timestamp", data->timestamp);
00055
00056     // Convert to string
00057     char *json_str = cJSON_PrintUnformatted(root);
00058     cJSON_Delete(root);
00059
00060     return json_str;
00061 }

```

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

<i>data</i>	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

<code>date_time</code>	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.](#)

```
00001 #ifndef COMMON_H
00002 #define COMMON_H
00003
00004 #include "dht_manager.h"
00005 #include "esp_sntp.h"
00006 #include "cJSON.h"
00007 #include "esp_log.h"
00008
00009 #define MEASURE_INTERVAL 60 * 1000
00010 #define WIFI_RECONNECT_INTERVAL_MS 60 * 1000
00011 #define TIMER_ID 1
00012 #define STACK_SIZE 4 * 1024
00013 #define MAX_Q_SIZE 10
00014
00022 void setup_sntp(void);
00023
00035 void get_current_date_time(char *date_time);
00036
00050 char *create_json_payload(const dht_data_t *data);
00051
00052 #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.](#)

```
00001 #ifndef DHT_SENSOR_H
00002 #define DHT_SENSOR_H
00003
00004 #include "dht.h"
00005
00006 #define CONFIG_EXAMPLE_INTERNAL_PULLUP 0
00007 #define CONFIG_EXAMPLE_TYPE_DHT11 0
00008 #define ISO8601_STR_LEN 25 // "YYYY-MM-DDTHH:MM:SSZ" + null
00009 #define DHT_SENSOR_TYPE (CONFIG_EXAMPLE_TYPE_AM2301 ? DHT_TYPE_AM2301 : CONFIG_EXAMPLE_TYPE_DHT11 ?
    DHT_TYPE_DHT11 : DHT_TYPE_SI7021)
00010
00017 typedef struct
00018 {
00019     float temperature;
00020     float humidity;
00021     char timestamp[ISO8601_STR_LEN];
00022 } dht_data_t;
00023
00034 esp_err_t setup_dht(void);
00035
00036 #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

<i>u8g2</i>	The u8g2 display structure initialized by init_oled_display()
<i>date</i>	Pointer to a null-terminated string containing the date information
<i>time</i>	Pointer to a null-terminated string containing the time information
<i>temp</i>	Pointer to a null-terminated string containing the temperature reading
<i>hum</i>	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.](#)

```

00001 #include "display_manager.h"
00002
00003 u8g2_t init_oled_display(void)
00004 {
00005     static const char *TAG = "oled_display_module";
00006     /* OLED Display Setup*/
00007     u8g2_t u8g2;
00008     u8g2_esp32_hal_t u8g2_esp32_hal = U8G2_ESP32_HAL_DEFAULT;
00009     u8g2_esp32_hal.bus.i2c.scl = I2C_SCL;
00010     u8g2_esp32_hal.bus.i2c.sda = I2C_SDA;
00011     u8g2_esp32_hal_init(u8g2_esp32_hal);
00012
00013     u8g2_Setup_ssd1306_i2c_128x64_noname_f(&u8g2, U8G2_R0,
00014                                           // u8x8_byte_sw_i2c,
00015                                           u8g2_esp32_i2c_byte_cb,
00016                                           u8g2_esp32_gpio_and_delay_cb);
00017     u8x8_SetI2CAddress(&u8g2.u8x8, 0x78);
00018     ESP_LOGI(TAG, "Init Display");
00019     u8g2_InitDisplay(&u8g2); // send init sequence to the display, display is in
00020                             // sleep mode after this,
00021
00022     ESP_LOGI(TAG, "WakeUp Display");
00023     u8g2_SetPowerSave(&u8g2, 0); // wake up display
00024     ESP_LOGI(TAG, "Clear Buffer");
00025
00026     u8g2_SetFont(&u8g2, u8g2_font_unifont_t_symbols);
00027
00028     u8g2_ClearBuffer(&u8g2);
00029     /* End of setup*/
00030
00031     return u8g2;
00032 }
00033
00034 void show_dht_data(u8g2_t u8g2, const char *date, const char *time, const char *temp, const char *hum)
00035 {
00036     u8g2_ClearBuffer(&u8g2);
00037     u8g2_DrawStr(&u8g2, 0, 12, date);
00038     u8g2_DrawStr(&u8g2, 0, 28, time);
00039     u8g2_DrawStr(&u8g2, 0, 46, temp);
00040     u8g2_DrawGlyph(&u8g2, 110, 46, 0x2600);
00041     u8g2_DrawStr(&u8g2, 0, 62, hum);
00042     u8g2_DrawGlyph(&u8g2, 110, 62, 0x2614);
00043     u8g2_SendBuffer(&u8g2); // Send the buffer data to display
00044 }

```

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

<code>u8g2</code>	The u8g2 display structure initialized by <code>init_oled_display()</code>
<code>date</code>	Pointer to a null-terminated string containing the date information
<code>time</code>	Pointer to a null-terminated string containing the time information
<code>temp</code>	Pointer to a null-terminated string containing the temperature reading
<code>hum</code>	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.](#)

```

00001 #ifndef DISPLAY_H
00002 #define DISPLAY_H
00003
00004 #include "driver/i2c_master.h"
00005 #include "u8g2.h"
00006 #include "u8g2_esp32_hal.h"
00007 #include "esp_log.h"
00008
00009 #define I2C_SDA 8
00010 #define I2C_SCL 9
00011
00023 u8g2_t init_oled_display(void);
00024
00039 void show_dht_data(u8g2_t u8g2, const char *date, const char *time, const char *temp, const char
    *hum);
00040
00041 #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.](#)

```

00001 #include "mqtt_manager.h"
00002
00003
00004 bool MQTT_CONNECTED = false;
00005 esp_mqtt_client_handle_t client = NULL;
00006
00007 static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void
    *event_data)
00008 {
00009     const char *TAG = "MQTT_EVENT_HANDLER";
00010     esp_mqtt_event_handle_t event = event_data;
00011     esp_mqtt_client_handle_t client = event->client;
00012     int msg_id;
00013     switch ((esp_mqtt_event_id_t)event_id)
00014     {
00015     case MQTT_EVENT_CONNECTED:
00016         ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
00017         MQTT_CONNECTED = true;
00018
00019         msg_id = esp_mqtt_client_subscribe(client, "/home/office/dht", 0);
00020         ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);
00021         break;
00022     case MQTT_EVENT_DISCONNECTED:
00023         ESP_LOGW(TAG, "MQTT_EVENT_DISCONNECTED");
00024         MQTT_CONNECTED = false;
00025         break;
00026
00027     case MQTT_EVENT_SUBSCRIBED:
00028         ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
00029         break;
00030     case MQTT_EVENT_UNSUBSCRIBED:
00031         ESP_LOGW(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
00032         break;
00033     case MQTT_EVENT_PUBLISHED:
00034         ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
00035         break;
00036     case MQTT_EVENT_DATA:
00037         ESP_LOGI(TAG, "TOPIC=.%s", event->topic_len, event->topic);
00038         ESP_LOGI(TAG, "DATA=.%s", event->data_len, event->data);
00039         break;
00040     case MQTT_EVENT_ERROR:
00041         ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
00042         break;
00043     default:
00044         ESP_LOGI(TAG, "Other event id:%d", event->event_id);
00045         break;
00046     }
00047 }
00048
00049 void setup_mqtt(void)
00050 {
00051     const char *TAG = "Setup MQTT";
00052     ESP_LOGI(TAG, "STARTING MQTT");
00053     esp_mqtt_client_config_t mqttConfig = {
00054         .broker.address.uri = CONFIG_BROKER_URI,
00055     };
00056
00057     client = esp_mqtt_client_init(&mqttConfig);
00058     esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
00059     esp_mqtt_client_start(client);
00060 }

```

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.](#)

```
00001 #ifndef MQTT_MANAGER_H
00002 #define MQTT_MANAGER_H
00003
00004 #include "mqtt_client.h"
00005 #include "esp_log.h"
00006
00017 void setup_mqtt(void);
00018
00019 #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.](#)

```

00001 #include "wifi_manager.h"
00002
00003 /* FreeRTOS event group to signal when we are connected*/
00004 EventGroupHandle_t s_wifi_event_group;
00005
00006 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00007                               int32_t event_id, void *event_data)
00008 {
00009     const char *TAG = "Wifi Event Handler";
00010     if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
00011     {
00012         esp_wifi_connect();
00013     }
00014     else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
00015     {
00016         ESP_LOGI(TAG, "Disconnected from AP, will try to reconnect in 60 seconds");
00017         xEventGroupClearBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00018     }
00019     else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
00020     {
00021         ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
00022         ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
00023         // initialize_sntp();
00024         // mqtt_app_start();
00025         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00026     }
00027 }
00028 void setup_wifi(void)
00029 {
00030     const char *TAG = "Setup Wifi";
00031     ESP_LOGI(TAG, "Setting up WiFi");
00032     s_wifi_event_group = xEventGroupCreate();
00033
00034     ESP_ERROR_CHECK(esp_netif_init());
00035
00036     ESP_ERROR_CHECK(esp_event_loop_create_default());
00037     esp_netif_create_default_wifi_sta();
00038
00039     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00040     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00041
00042     esp_event_handler_instance_t instance_any_id;
00043     esp_event_handler_instance_t instance_got_ip;
00044     ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
00045                                                         ESP_EVENT_ANY_ID,
00046                                                         &wifi_event_handler,
00047                                                         NULL,
00048                                                         &instance_any_id));
00049     ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
00050                                                         IP_EVENT_STA_GOT_IP,
00051                                                         &wifi_event_handler,
00052                                                         NULL,
00053                                                         &instance_got_ip));
00054
00055     wifi_config_t wifi_config = {
00056         .sta = {
00057             .ssid = EXAMPLE_ESP_WIFI_SSID,
00058             .password = EXAMPLE_ESP_WIFI_PASS,
00059             /* Authmode threshold resets to WPA2 as default if password matches WPA2 standards
00060              * If you want to connect the device to deprecated WEP/WPA networks, Please set the
00061              * threshold value
00062              * to WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK and set the password with length and format matching
00063              * to
00064              * WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK standards.
00065              */
00066             .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
00067             .sae_pwe_h2e = ESP_WIFI_SAE_MODE,
00068             .sae_h2e_identifier = EXAMPLE_H2E_IDENTIFIER,
00069         },
00070     };
00071     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00072     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
00073     ESP_ERROR_CHECK(esp_wifi_start());
00074     ESP_LOGI(TAG, "wifi_task started and wifi driver started");
00075 }

```

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.](#)

```
00001 #ifndef WIFI_MANAGER_H
00002 #define WIFI_MANAGER_H
00003
00004 #include "esp_log.h"
00005 #include "esp_wifi.h"
00006 #include "freertos/event_groups.h"
00007 #include "freertos/FreeRTOS.h"
00008
00009 #define EXAMPLE_ESP_WIFI_SSID CONFIG_ESP_WIFI_SSID
00010 #define EXAMPLE_ESP_WIFI_PASS CONFIG_ESP_WIFI_PASSWORD
00011 #define EXAMPLE_ESP_MAXIMUM_RETRY CONFIG_ESP_MAXIMUM_RETRY
00012
00013 #if CONFIG_ESP_WPA3_SAE_PWE_HUNT_AND_PECK
00014 #define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HUNT_AND_PECK
00015 #define EXAMPLE_H2E_IDENTIFIER ""
00016 #elif CONFIG_ESP_WPA3_SAE_PWE_HASH_TO_ELEMENT
00017 #define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HASH_TO_ELEMENT
00018 #define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
00019 #elif CONFIG_ESP_WPA3_SAE_PWE_BOTH
00020 #define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_BOTH
00021 #define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
00022 #endif
00023 #if CONFIG_ESP_WIFI_AUTH_OPEN
00024 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
00025 #elif CONFIG_ESP_WIFI_AUTH_WEP
00026 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
00027 #elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
00028 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
00029 #elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
00030 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
00031 #elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
00032 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_WPA2_PSK
00033 #elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
00034 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
00035 #elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
00036 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_WPA3_PSK
00037 #elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
```

```

00038 #define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
00039 #endif
00040
00041 /* The event group allows multiple bits for each event, but we only care about two events:
00042  * - we are connected to the AP with an IP
00043  * - we failed to connect after the maximum amount of retries */
00044 #define WIFI_CONNECTED_BIT BIT0
00045
00062 void setup_wifi(void);
00063
00064 #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

<i>timer</i>	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

<code>args</code>	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

<i>args</i>	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

<i>args</i>	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.](#)

```

00001
00015
00016 #include <stdio.h>
00017 #include <string.h>
00018 #include <time.h>
00019
00020 #include "dht_manager.h"
00021 #include "display_manager.h"
00022 #include "wifi_manager.h"
00023 #include "mqtt_manager.h"
00024 #include "common.h"
00025
00026 #include "freertos/task.h"
00027 #include "freertos/timers.h"
00028 #include "freertos/queue.h"
00029
00030 #include "esp_system.h"
00031 #include "esp_event.h"
00032 #include "nvs_flash.h"
00033
00034 extern EventGroupHandle_t s_wifi_event_group;
00035 extern bool MQTT_CONNECTED;
00036 extern esp_mqtt_client_handle_t client;
00037
00038 TimerHandle_t timerDHT;
00039 QueueHandle_t displayQueue;
00040 QueueHandle_t mqttQueue;
00041
00042 static const char *TAG = "iot_env_station";
00043
00044 /* Function prototypes */
00045 esp_err_t setup_timer(void);
00046 esp_err_t create_tasks(void);
00047
00048 void measure_temp_hum(TimerHandle_t timer);
00049 void task_show_data_oled(void *args);
00050 void task_send_data_mqtt(void *args);
00051 void task_wifi(void *args);
00052
00067 void app_main(void)
00068 {
00069     // Initialize NVS
00070     esp_err_t ret = nvs_flash_init();
00071     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
00072     {
00073         ESP_ERROR_CHECK(nvs_flash_erase());
00074         ret = nvs_flash_init();
00075     }
00076     ESP_ERROR_CHECK(ret);
00077
00078     displayQueue = xQueueCreate(MAX_Q_SIZE, sizeof(dht_data_t));
00079     mqttQueue = xQueueCreate(MAX_Q_SIZE, sizeof(dht_data_t));
00080     ESP_ERROR_CHECK(setup_dht());
00081     ESP_ERROR_CHECK(setup_timer());
00082     ESP_ERROR_CHECK(create_tasks());
00083 }
00084
00096 esp_err_t setup_timer(void)
00097 {
00098     timerDHT = xTimerCreate("Timer DHT",
00099                             pdMS_TO_TICKS(MEASURE_INTERVAL),
00100                             pdTRUE,
00101                             (void *)TIMER_ID,
00102                             measure_temp_hum);
00103     if (timerDHT == NULL)
00104     {
00105         ESP_LOGE(TAG, "Timer could not be created");
00106         return ESP_FAIL;
00107     }
00108     else
00109     {
00110         if (xTimerStart(timerDHT, 0) != pdPASS)
00111         {
00112             ESP_LOGE(TAG, "The timer could not be set into the Active state");
00113             return ESP_FAIL;
00114         }
00115     }
00116     return ESP_OK;
00117 }
00118
00133 esp_err_t create_tasks(void)
00134 {

```

```

00135     static uint8_t ucParameterToPass;
00136     TaskHandle_t displayHandle = NULL;
00137     TaskHandle_t mqttHandle = NULL;
00138     TaskHandle_t wifiHandle = NULL;
00139
00140     if (xTaskCreate(task_show_data_oled,
00141                   "Show read data on oled display",
00142                   STACK_SIZE,
00143                   &ucParameterToPass,
00144                   1,
00145                   &displayHandle) != pdPASS)
00146     {
00147         return ESP_FAIL;
00148     }
00149
00150     if (xTaskCreate(task_wifi,
00151                   "Task to connect to wifi",
00152                   STACK_SIZE,
00153                   &ucParameterToPass,
00154                   3,
00155                   &wifiHandle) != pdPASS)
00156     {
00157         return ESP_FAIL;
00158     }
00159
00160     if (xTaskCreate(task_send_data_mqtt,
00161                   "Send data to MQTT broker",
00162                   STACK_SIZE,
00163                   &ucParameterToPass,
00164                   2,
00165                   &mqttHandle) != pdPASS)
00166     {
00167         return ESP_FAIL;
00168     }
00169     return ESP_OK;
00170 }
00171
00172 void task_show_data_oled(void *args)
00173 {
00174     dht_data_t sensorData = {0};
00175     u8g2_t u8g2;
00176     // OLED Display Setup
00177     u8g2 = init_oled_display();
00178
00179     while (true)
00180     {
00181         if (xQueueReceive(displayQueue, &sensorData, portMAX_DELAY))
00182         {
00183             struct tm timeinfo;
00184             char temp_str[20], hum_str[20], date_str[17], time_str[12];
00185             // Parse ISO8601 string back to time
00186             strptime(sensorData.timestamp, "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
00187             // Format the date as "dd/mm/YYYY"
00188             strftime(date_str, sizeof(date_str), "Date: %d/%m/%Y", &timeinfo);
00189             // Format the time as "HH:mm"
00190             strftime(time_str, sizeof(time_str), "Time: %H:%M", &timeinfo);
00191             sprintf(temp_str, "Temp: %.2fC", sensorData.temperature);
00192             sprintf(hum_str, "Hum: %.2f %", sensorData.humidity);
00193             show_dht_data(u8g2, date_str, time_str, temp_str, hum_str);
00194         }
00195         else
00196         {
00197             ESP_LOGE(TAG, "Error receiving data or no data in buffer");
00198             vTaskDelay(pdMS_TO_TICKS(100));
00199         }
00200     }
00201 }
00202
00203 void task_send_data_mqtt(void *args)
00204 {
00205     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00206                                           WIFI_CONNECTED_BIT,
00207                                           pdFALSE,
00208                                           pdTRUE,
00209                                           pdMS_TO_TICKS(portMAX_DELAY));
00210
00211     if ((bits & WIFI_CONNECTED_BIT) == 1)
00212     {
00213         ESP_LOGI(TAG, "WiFi connected, starting MQTT task and SNTP");
00214         setup_mqtt();
00215         setup_sntp();
00216     }
00217
00218     dht_data_t sensorData = {0};
00219     while (true)
00220     {

```

```
00252         if ((xQueueReceive(mqttQueue, &sensorData, portMAX_DELAY)) && MQTT_CONNECTED)
00253         {
00254             // Convert to JSON string
00255             char *json_str = create_json_payload(&sensorData);
00256             esp_mqtt_client_publish(client, "/home/office/dht", json_str, 0, 0, 0);
00257             free(json_str);
00258         }
00259         else
00260         {
00261             ESP_LOGE(TAG, "Error receiving data or no data in buffer");
00262         }
00263         vTaskDelay(pdMS_TO_TICKS(100));
00264     }
00265 }
00266
00282 void task_wifi(void *args)
00283 {
00284     setup_wifi();
00285
00286     while (true)
00287     {
00288         EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00289                                                 WIFI_CONNECTED_BIT,
00290                                                 pdFALSE,
00291                                                 pdFALSE,
00292                                                 pdMS_TO_TICKS(WIFI_RECONNECT_INTERVAL_MS));
00293
00294         if ((bits & WIFI_CONNECTED_BIT) == 0)
00295         {
00296             esp_wifi_connect();
00297         }
00298         vTaskDelay(pdMS_TO_TICKS(1000));
00299     }
00300 }
00301
00318 void measure_temp_hum(TimerHandle_t timer)
00319 {
00320     esp_err_t res;
00321     dht_data_t dhtData;
00322     res = dht_read_float_data(DHT_SENSOR_TYPE, CONFIG_ESP_TEMP_SENSOR_GPIO, &dhtData.humidity,
00323                               &dhtData.temperature);
00324     get_current_date_time(dhtData.timestamp);
00325     if (res == ESP_OK)
00326     {
00327         if (xQueueSend(displayQueue, &dhtData, pdMS_TO_TICKS(100)) != pdPASS)
00328         {
00329             ESP_LOGE(TAG, "Error sending data to display queue");
00330         }
00331         if (xQueueSend(mqttQueue, &dhtData, pdMS_TO_TICKS(100)) != pdPASS)
00332         {
00333             ESP_LOGE(TAG, "Error sending data to MQTT queue");
00334         }
00335     }
00336     else
00337     {
00338         ESP_LOGE(TAG, "Error reading data");
00339     }
00340 }
```


Index

- app_main
 - main.c, [27](#)
- client
 - main.c, [30](#)
 - mqtt_manager.c, [19](#)
- common.c
 - create_json_payload, [7](#)
 - get_current_date_time, [7](#)
 - setup_snmp, [8](#)
 - time_sync_notification_cb, [8](#)
- common.h
 - create_json_payload, [10](#)
 - get_current_date_time, [10](#)
 - MAX_Q_SIZE, [10](#)
 - MEASURE_INTERVAL, [10](#)
 - setup_snmp, [11](#)
 - STACK_SIZE, [10](#)
 - TIMER_ID, [10](#)
 - WIFI_RECONNECT_INTERVAL_MS, [10](#)
- CONFIG_EXAMPLE_INTERNAL_PULLUP
 - dht_manager.h, [13](#)
- CONFIG_EXAMPLE_TYPE_DHT11
 - dht_manager.h, [13](#)
- create_json_payload
 - common.c, [7](#)
 - common.h, [10](#)
- create_tasks
 - main.c, [27](#)
- dht_data_t, [5](#)
 - humidity, [5](#)
 - temperature, [5](#)
 - timestamp, [6](#)
- dht_manager.c
 - setup_dht, [12](#)
- dht_manager.h
 - CONFIG_EXAMPLE_INTERNAL_PULLUP, [13](#)
 - CONFIG_EXAMPLE_TYPE_DHT11, [13](#)
 - DHT_SENSOR_TYPE, [13](#)
 - ISO8601_STR_LEN, [13](#)
 - setup_dht, [14](#)
- DHT_SENSOR_TYPE
 - dht_manager.h, [13](#)
- display_manager.c
 - init_oled_display, [15](#)
 - show_dht_data, [15](#)
- display_manager.h
 - I2C_SCL, [17](#)
 - I2C_SDA, [17](#)
- init_oled_display
 - display_manager.h, [17](#)
- show_dht_data, [17](#)
- displayQueue
 - main.c, [30](#)
- EXAMPLE_ESP_MAXIMUM_RETRY
 - wifi_manager.h, [24](#)
- EXAMPLE_ESP_WIFI_PASS
 - wifi_manager.h, [24](#)
- EXAMPLE_ESP_WIFI_SSID
 - wifi_manager.h, [24](#)
- get_current_date_time
 - common.c, [7](#)
 - common.h, [10](#)
- humidity
 - dht_data_t, [5](#)
- I2C_SCL
 - display_manager.h, [17](#)
- I2C_SDA
 - display_manager.h, [17](#)
- includes/common.c, [7](#), [8](#)
- includes/common.h, [9](#), [11](#)
- includes/dht_manager.c, [11](#), [12](#)
- includes/dht_manager.h, [12](#), [14](#)
- includes/display_manager.c, [14](#), [16](#)
- includes/display_manager.h, [16](#), [18](#)
- includes/mqtt_manager.c, [18](#), [20](#)
- includes/mqtt_manager.h, [20](#), [21](#)
- includes/wifi_manager.c, [21](#), [23](#)
- includes/wifi_manager.h, [24](#), [25](#)
- init_oled_display
 - display_manager.c, [15](#)
 - display_manager.h, [17](#)
- ISO8601_STR_LEN
 - dht_manager.h, [13](#)
- main.c
 - app_main, [27](#)
 - client, [30](#)
 - create_tasks, [27](#)
 - displayQueue, [30](#)
 - measure_temp_hum, [28](#)
 - MQTT_CONNECTED, [30](#)
 - mqttQueue, [31](#)
 - s_wifi_event_group, [31](#)
 - setup_timer, [28](#)
 - TAG, [31](#)
 - task_send_data_mqtt, [29](#)

- task_show_data_oled, [29](#)
- task_wifi, [30](#)
- timerDHT, [31](#)
- main/main.c, [26](#), [32](#)
- MAX_Q_SIZE
 - common.h, [10](#)
- MEASURE_INTERVAL
 - common.h, [10](#)
- measure_temp_hum
 - main.c, [28](#)
- MQTT_CONNECTED
 - main.c, [30](#)
 - mqtt_manager.c, [19](#)
- mqtt_event_handler
 - mqtt_manager.c, [19](#)
- mqtt_manager.c
 - client, [19](#)
 - MQTT_CONNECTED, [19](#)
 - mqtt_event_handler, [19](#)
 - setup_mqtt, [19](#)
- mqtt_manager.h
 - setup_mqtt, [21](#)
- mqttQueue
 - main.c, [31](#)
- s_wifi_event_group
 - main.c, [31](#)
 - wifi_manager.c, [22](#)
- setup_dht
 - dht_manager.c, [12](#)
 - dht_manager.h, [14](#)
- setup_mqtt
 - mqtt_manager.c, [19](#)
 - mqtt_manager.h, [21](#)
- setup_snmp
 - common.c, [8](#)
 - common.h, [11](#)
- setup_timer
 - main.c, [28](#)
- setup_wifi
 - wifi_manager.c, [22](#)
 - wifi_manager.h, [25](#)
- show_dht_data
 - display_manager.c, [15](#)
 - display_manager.h, [17](#)
- STACK_SIZE
 - common.h, [10](#)
- TAG
 - main.c, [31](#)
- task_send_data_mqtt
 - main.c, [29](#)
- task_show_data_oled
 - main.c, [29](#)
- task_wifi
 - main.c, [30](#)
- temperature
 - dht_data_t, [5](#)
- time_sync_notification_cb
 - common.c, [8](#)
- TIMER_ID
 - common.h, [10](#)
- timerDHT
 - main.c, [31](#)
- timestamp
 - dht_data_t, [6](#)
- WIFI_CONNECTED_BIT
 - wifi_manager.h, [24](#)
- wifi_event_handler
 - wifi_manager.c, [22](#)
- wifi_manager.c
 - s_wifi_event_group, [22](#)
 - setup_wifi, [22](#)
 - wifi_event_handler, [22](#)
- wifi_manager.h
 - EXAMPLE_ESP_MAXIMUM_RETRY, [24](#)
 - EXAMPLE_ESP_WIFI_PASS, [24](#)
 - EXAMPLE_ESP_WIFI_SSID, [24](#)
 - setup_wifi, [25](#)
 - WIFI_CONNECTED_BIT, [24](#)
- WIFI_RECONNECT_INTERVAL_MS
 - common.h, [10](#)