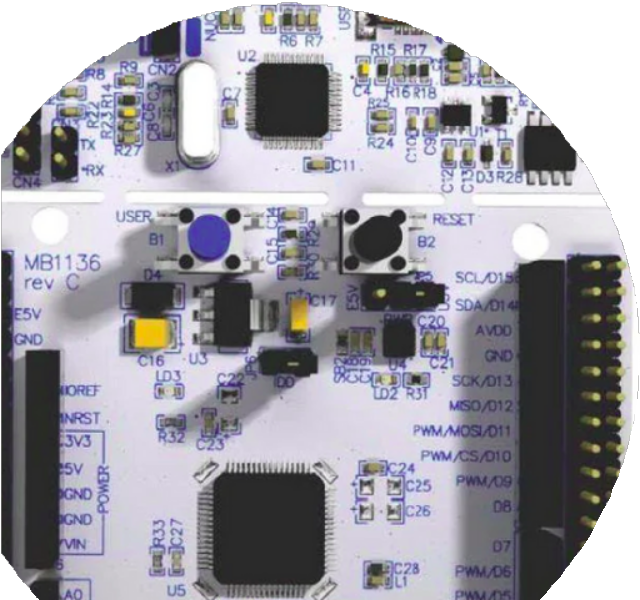
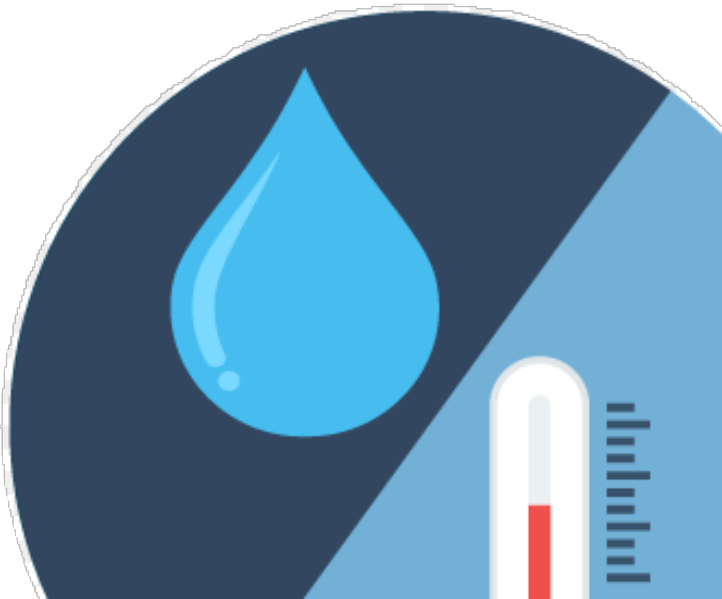
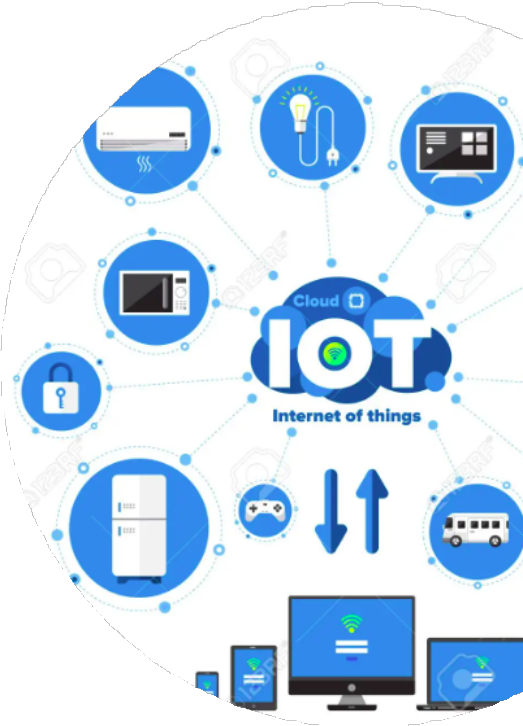
IoT-based Distance measurement System



**PRESENTED BY**

**HARISH KUMAR A**

**NOVEMBER-2023**

# Table of Contents

**Project Summary**

**1**

**page no:**

**01**

**key features of the project** **03**

**2**

**Hardware Used and there Specification** **04**

**3**

**Finite state machine implementation** **05**

**4**

**block diagram for finite state machine** **06**

**4.1**

**Why interrupt based uart communication** **08**

**5**

**Connection diagram** **10**

**6**

**AHT25 Sens** **11**

**7**

**W10 wifi module** **15**

**8**

**code snippet to wifi module initialization and**

**9**

**connection with interrupt-based uart commands** **16**

**code snippet to MQTT initialization and**

**10**

**18**

**connection with interrupt-based uart commands**

**FreeRTOS Task and Queues** **20**

**11**

**Project code** **26**

**12**

**Rightech IoT Cloud** **32**

**13**

**JSON (JavaScript Object Notation)** **33**

**14**

**JSON code** **35**

**15**

**1**

1. **Project Summary**

The Internet of Things (IoT) is a rapidly growing field that has the potential to revolutionize the way we interact with the world around us. One of the most promising applications of IoT is in the area of environmental monitoring. By deploying IoT devices in various environments, we can collect real-time data on environmental conditions such as measuring. This data can then be used to track changes in environmental conditions over time, identify potential problems, and take corrective action.

This project proposes the development of an IoT-based temperature and humidity monitoring system using STM32, AHT25, and W10 WiFi modules. The system will use an STM32 microcontroller to acquire sensor data from an AHT25 temperature and humidity sensor. The sensor data will then be published to MQTT using the W10 WiFi module, where it can be accessed by a web application or other IoT devices. The web application will allow users to visualize the sensor data and track changes in environmental conditions over time.

The system will be deployed in a real-world environment to test its performance. The deployment results will be used to assess the system's feasibility and identify any potential improvements.

The proposed system has the potential to be a valuable tool for tracking and monitoring environmental conditions. The system is relatively low-cost and easy to deploy, making it a feasible option for a wide range of applications. The system is also scalable, making it possible to deploy it in a variety of environments.

The results of this project will contribute to the body of knowledge on IoT- based environmental monitoring systems. The project will also provide a valuable case study for other researchers and developers who are interested in developing IoT-based ecological monitoring systems.

**2**

1. **key features of the project**

Sensor Initialization with State Machine and Application state Machine maintained.

UART Communication is based on Interrupt.

Common function for UART receive data parsing.

Timer based delay.

STM32 Application is Master and configured with W10 (Wi-Fi) module accordingly with AT commands.

FreeRTOS Task with priority implimantation.

QUEUE are used for inter proces communication.

Data is pushed to the MQTT server.

Data is in JASON format.

Temperature and Humidity data is pushed in every 1 Min.

**3**

## Hardware Used and there Specification

###### STM32F411RE microcontroller

The STM32F411RE is a high-performance microcontroller based on the ARM Cortex- M4 processor. It has a number of features that make it well-suited for a variety of applications, including:

 100 MHz CPU clock speed  128 KB of RAM

 512 KB of Flash memory  Floating point unit (FPU)

 11 general-purpose timers

 13 communication interfaces  USB OTG

 RTC

###### AHT25 Humidity and Temperature Module

* + Relative humidity and temperature output
  + Superior sensor performance, typical accuracy RH: ±2%, T: ±0.3℃
  + Fully calibrated and processed digital output, I²C protocol
  + Wide voltage support 2.2 to 5.5V DC
  + Excellent long-term stability
  + Fast-response and anti-interference capability

###### W10 WiFi

The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

 100mW transmit power  11Mbps data rate

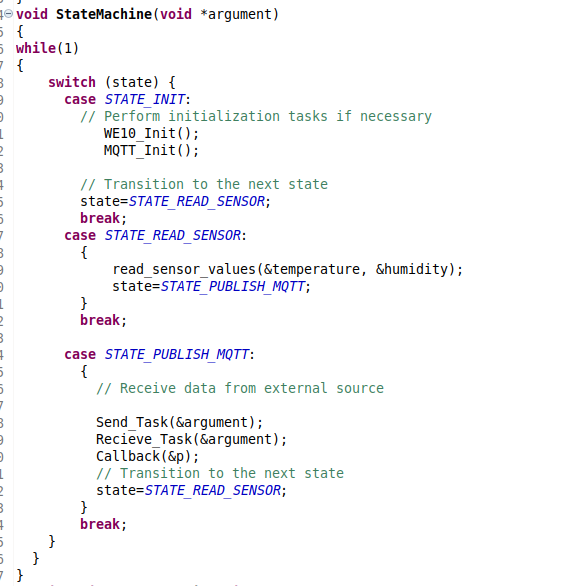
802.11 b/g/n compatibility  Integrated antenna

**4**

## Finite state machine implementation

This finite state machine has three states: STATE\_INIT, STATE\_READ\_SENSOR, and STATE\_PUBLISH\_MQTT. The STATE\_INIT

state is the initial state, and it is where the sensor data is read. The STATE\_READ\_SENSOR state is where the sensor data is measured, and the PUBLISH state is where the sensor data is published to MQTT. The Callback in every 1 minute, and it updates the state of the finite state machine.



**5**

**4.1. block diagram for finite state machine**

**PUBLISH state**

**\* Send\_Task()**

**\* Recieve\_Task()**

**MEASURE state**

**\* read\_sensor\_values()**

**INIT state**

**WE10\_Init()**

**MQTT\_Init()**

**Figure.1**

The block diagram shows the different states of the finite state machine and the actions that are performed in each state. The arrows represent the transitions between the states.

The Start block represents the beginning of the code. The State Machine block represents the main loop of the code. The STATE\_INIT block represents the initialization state of the machine. The STATE\_READ\_SENSOR block represents the state where the sensor values are read and published to MQTT. The STATE\_PUBLISH\_MQTT block represents the state where data is sent and received from an external source. The End block represents the end of the code.

**6**

1. The StateMachine() function is called.
2. The switch() statement checks the current state of the machine.
3. If the current state is STATE\_INIT, the WE10\_Init() and MQTT\_Init() functions are called to initialize the sensor and the MQTT connection.
4. The state is then changed to STATE\_PUBLISH\_MQTT.
5. If the current state is STATE\_READ\_SENSOR, the read\_sensor\_values() function is called to read the sensor values.
6. The state is then changed to STATE\_PUBLISH\_MQTT.
7. If the current state is STATE\_PUBLISH\_MQTT, the Send\_Task() and Recieve\_Task() functions are called to send and receive data from the external source.
8. The state is then changed to STATE\_READ\_SENSOR.
9. The while loop is repeated.

The StateMachine() function will continue to run until it is terminated. The flow of the code will depend on the current state of the machine.

Here is a more detailed explanation of the flow of the code:

 In the STATE\_INIT state, the WE10\_Init() and MQTT\_Init() functions are called to initialize the sensor and the MQTT connection.

 In the STATE\_READ\_SENSOR state, the read\_sensor\_values() function is called to read the sensor values. The sensor values are then published to MQTT.

 In the STATE\_PUBLISH\_MQTT state, the Send\_Task() and Recieve\_Task() functions are called to send and receive data from the external source.

 The while loop is repeated until the StateMachine() function is terminated.

**7**

## Why interrupt based uart communication

The advantages of interrupt-based UART communication over other methods:

 Improved efficiency: The CPU is not constantly polling the UART status, so it can be used for other tasks. This can improve the efficiency of the system and reduce power consumption.

 Reduced latency: Interrupt-based communication can be faster than polling, as the CPU is only interrupted when data is ready to be sent or received. This can improve the responsiveness of the system.

 Better multitasking: Interrupt-based communication allows the CPU to handle multiple UART events simultaneously. This can be useful for applications that require frequent and asynchronous serial communication.

Here are some of the disadvantages of interrupt-based UART communication:

 More complex code: Interrupt-based communication is more complex to implement than polling. This is because the programmer needs to write code to handle the interrupts.

 More overhead: Interrupt-based communication can have more overhead than polling. This is because the CPU needs to save and restore its state when it is interrupted.

Overall, interrupt-based UART communication is a more efficient and responsive way to communicate with serial devices. However, it is more complex to implement and can have more overhead.

**8**

Here are some examples of applications where interrupt-based UART communication would be a good choice:

 **Sensor networks:** Sensor networks often need to communicate with each other or with a central server. Interrupt-based communication can be used to improve the efficiency and responsiveness of these networks.

 **Wireless modules:** Wireless modules often use UART to communicate with the host microcontroller. Interrupt-based communication can be used to improve the performance of these modules.

 **Real-time systems:** Real-time systems often need to communicate with other devices in a timely manner. Interrupt-based communication can be used to ensure that these communications are not missed.

**9**

6. Connection diagram

**COLOUR**

Red Black purple Pink yellow blue



**AHT25**

**W10**

**stm32f411re**

**Figure.2**

**PIN**

**NUMBER**

vcc GND PB8 PB9 PA9 PA10

Pin configuration

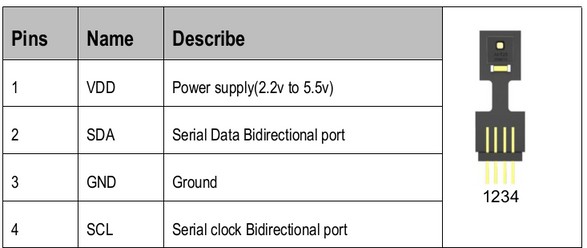
**10**

|  |  |  |  |
| --- | --- | --- | --- |
| **PIN** |  |  | **COMPONENTS** |
| vcc 5v |  |  | w10 and AHT25 |
| ground | w10 and AHT25 |
| SCL | AHT25 |
| SDK | AHT25 |
| Tx | W10 |
| Rx | W10 |

### 7. AHT25 Sensor

**Product Description :**

The AHT25 can be widely used in consumer electronics, medical, automotive, industrial, meteorological, and other fields, such as HVAC, dehumidifiers and refrigerators, and other home appliances, testing and testing equipment, and other related temperature and humidity testing and control products.



Note:

1. The power supply voltage of the host MCU must be consistent with the sensor when the product is used in the circuit.
2. If you need to further improve the reliability of the system, you can control the sensor power supply.
3. When the sensor is just powered on, give priority to the sensor VDD power supply, SCL and SDA high level can be set after

5ms.

To avoid signal conflicts, the microprocessor (MCU) must only drive SDA and SCL at low level. An external pull-up

resistor (for example: 4.7kΩ) is required to pull the signal to a high level. Refer to Table 7 and Table 8 for detailed

information about sensor input/output characteristics.

**11**

#### Slave address and configurations:

**AHT25 read address**

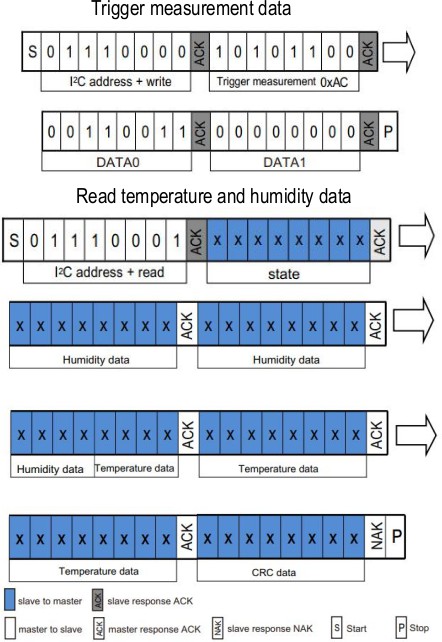
**0x70**

**AHT25 write address**

**0x71**

**AHT25 start measure address**

**0xAC**



**ILLIFRED**

**NDI**

**Signal Conversion :**

##### Relative Humidity Conversion :

The relative humidity RH can be calculated according to the relative humidity signal S RH output by SDA through the following formula

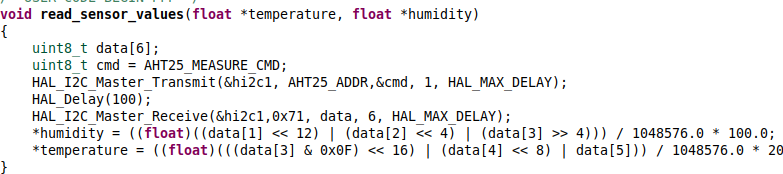
##### Temperature Conversion :

The temperature T can be calculated by substituting the temperature output signal S T into the following formula:

(The result is expressed in temperature ℃ ):



#### In c Code conversion :

The code first declares two variables, data and cmd. The data variable will be used to store the raw sensor data, and the cmd variable will be used to send the measurement command to the sensor.

The next few lines of code send the measurement command to the sensor and then wait 100 milliseconds for the sensor to complete the measurement.

The next line of code reads the raw sensor data into the data variable. The last two lines of code convert the raw sensor data into floating- point values and store them in the temperature and humidity variables.

The read\_sensor\_values() function is a simple example of how to read sensor data from an AHT25 sensor using the HAL I2C driver. The function takes two pointers to floating-point variables as input, and it stores the temperature and humidity values in these variables. The function returns void.

#### Signal Conversion :

\*humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4))) / 1048576.0 \* 100.0;

\*temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8)

| data[5])) / 1048576.0 \* 200.0 - 50.0;

The first line of code converts the raw humidity data from the sensor into a floating-point value. The raw humidity data is stored in the data[1], data[2], and data[3] bytes of the data array. The << and | operators are used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The

\* operator is then used to multiply the result by 100, which converts the value into a percentage.

The second line of code converts the raw temperature data from the sensor into a floating-point value. The raw temperature data is stored in the data[3], data[4], and data[5] bytes of the data array. The & operator is used to mask out the lower 4 bits of the data[3] byte. The

<< and | operators are then used to combine the three bytes into a single 32-bit integer. The / operator is then used to divide the integer by 1048576, which is the maximum value that can be stored in a 32-bit integer. The \* operator is then used to multiply the result by 200, which converts the value into degrees Celsius. The - 50.0 expression is then used to subtract 50 degrees Celsius from the result, which converts the value into degrees Fahrenheit.

### 8. W10 wifi module

****

The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built- in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:

 100mW transmit power  11Mbps data rate

802.11 b/g/n compatibility  Integrated antenna

**9.code snippet to wifi module initialization and connection with interrupt-based uart commands**

The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.

The next few lines of code send the CMD+RESET command to the WE10 module. This command resets the module to its default state.

The next line of code sends the CMD+WIFIMODE=1 command to the WE10 module. This command sets the module to operate in WiFi mode.

The next line of code sends the CMD+CONTOAP=SSID, PASSWD command to the WE10 module. This command configures the module to connect to the WiFi network with the specified SSID and password.

The next line of code sends the CMD.WIFI command to the WE10 module. This command queries the module for its WiFi status.

The last line of code waits for 2000 milliseconds and then receives a response from the WE10 module. The response is stored in the buffer.

The WE10\_Init() function is a simple example of how to initialize a WE10 module and connect it to a WiFi network. The function takes no arguments and it returns void.

**10.code snippet to MQTT initialization and connection with interrupt-based uart commands**

**N**

The code you provided is a initialize a WE10 module and connect it to an MQTT broker. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.

The next few lines of code send the CMD+MQTTNETCFG command to the WE10 module. This command configures the module to connect to the MQTT broker at dev.rightech.io on port 1883. The CMD+MQTTCONCFG command configures the module to connect to the MQTT broker as a client with the username mqtt-arifm4348- ud8eo8 and no password. The CMD+MQTTSTART command starts the MQTT client and connects to the broker. The CMD+MQTTSUB command subscribes the client to the topic base/relay/led1.

The MQTT\_Init() function is a simple example of how to initialize a WE10 module and connect it to an MQTT broker. The function takes no arguments and it returns void.

Here is a more detailed explanation of the code:

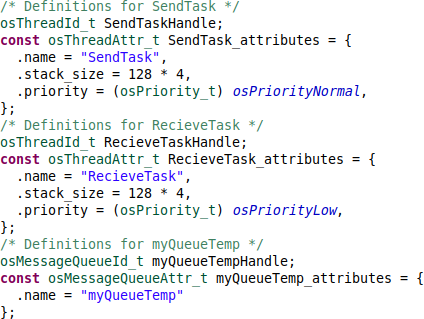
 The CMD+MQTTNETCFG command is used to configure the MQTT parameters of the WE10 module. The first parameter is the hostname or IP address of the MQTT broker. The second parameter is the port number of the MQTT broker.

 The CMD+MQTTCONCFG command is used to configure the MQTT client of the WE10 module. The first parameter is the username of the MQTT client. The second parameter is the password of the MQTT client.

 The CMD+MQTTSTART command is used to start the MQTT client of the WE10 module. This command connects the client to the MQTT broker.

 The CMD+MQTTSUB command is used to subscribe the MQTT client to a topic. The first parameter is the topic that the client wants to subscribe to.

**11.FreeRTOS Task and Queues**



###### SendTask:

The SendTask is a task that will be created by the code.

The osThreadId\_t SendTaskHandle variable is used to store the handle of the SendTask. The osThreadAttr\_t SendTask\_attributes structure defines the attributes of the SendTask.

The SendTask\_attributes structure has three members:  name: The name of the task.

 stack\_size The size of the stack that will be allocated to the task.  priority: The priority of the task.

In this case, the name of the task is "SendTask", the stack\_size is 128 \* 4 bytes, and the priority is osPriorityNormal.

The osThreadAttr\_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack\_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

###### RecieveTask:

The RecieveTask is a task that will be created by the code.

The osThreadId\_t RecieveTaskHandle variable is used to store the handle of the RecieveTask. The osThreadAttr\_t RecieveTask\_attributes structure defines the attributes of the RecieveTask.

The RecieveTask\_attributes structure has three members:  name: The name of the task.

 stack\_size The size of the stack that will be allocated to the task.  priority: The priority of the task.

In this case, the name of the task is "RecieveTask", the stack\_size is 128

\* 4 bytes, and the priority is osPriorityLow.

The osThreadAttr\_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack\_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

##### myQueueTemp:

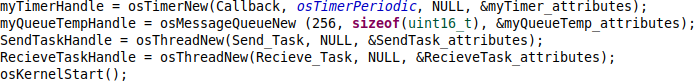
The myQueueTemp message queue will be created by the code.

The osMessageQueueId\_t myQueueTempHandle variable is used to store the handle of the myQueueTemp message queue. The osMessageQueueAttr\_t myQueueTemp\_attributes structure defines the attributes of the myQueueTemp message queue.

The myQueueTemp\_attributes structure has one member:  name: The name of the message queue.

In this case, the name of the message queue is "myQueueTemp".

The osMessageQueueAttr\_t structure is used to configure the attributes of a message queue. The name member is used to set the name of the message queue.



The code you provided creates the myQueueTemp message queue, creates the SendTask and RecieveTask tasks, and starts the kernel.

The myQueueTempHandle = osMessageQueueNew (256, sizeof(uint16\_t), &myQueueTemp\_attributes); line creates the myQueueTemp message queue. The osMessageQueueNew() function is used to create a message queue. The first parameter is the maximum number of messages that can be stored in the queue, the second parameter is the size of each message, and the third parameter is a pointer to the attributes of the queue.

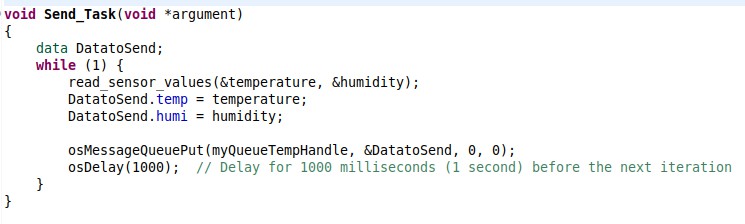
The SendTaskHandle = osThreadNew(Send\_Task, NULL, &SendTask\_attributes); line creates the SendTask task. The osThreadNew() function is used to create a task. The first parameter is the name of the task, the second parameter is a pointer to the function that will be executed by the task, and the third parameter is a pointer to the attributes of the task.

The RecieveTaskHandle = osThreadNew(Recieve\_Task, NULL, &RecieveTask\_attributes); line creates the RecieveTask task. The osThreadNew() function is used to create a task. The first parameter is the name of the task, the second parameter is a pointer to the function that will be executed by the task, and the third parameter is a pointer to the attributes of the task.

The osKernelStart(); line starts the kernel. The osKernelStart() function is used to start the kernel. The kernel is the core of the operating system. It is responsible for scheduling tasks, managing

resources, and handling interrupts.

**Send\_Task Function :**

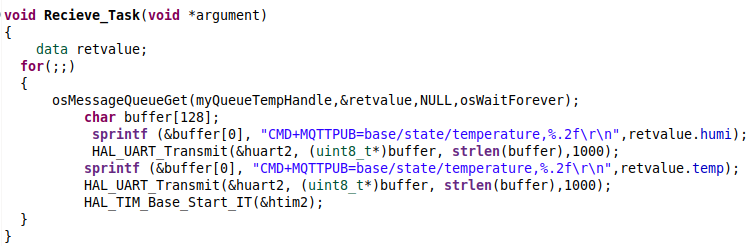
The code you provided defines the Send\_Task function. The Send\_Task is a task that will be executed by the code.

The Send\_Task function first declares a variable of type data called DatatoSend.

The temp member of the data structure is used to store the temperature reading, and the humidity member of the data structure is used to store the humidity reading.

The Send\_Task function then enters an infinite loop. In each iteration of the loop, the Send\_Task function reads the temperature and humidity readings from the sensors, stores the readings in the DatatoSend structure, and then puts the DatatoSend structure on the myQueueTemp message queue. The osMessageQueuePut() function is used to put a message on a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is the priority of the message, and the fourth parameter is the timeout value.

**Send\_Task Function :**



The code you provided defines the Recieve\_Task function. The Recieve\_Task is a task that will be executed by the code.

The Recieve\_Task function first declares a variable of type data called retvalue.

The temp member of the data structure is used to store the temperature reading, and the humi member of the data structure is used to store the humidity reading.

The Recieve\_Task function then enters an infinite loop. In each iteration of the loop, the Recieve\_Task function gets a message from the myQueueTemp message queue. The osMessageQueueGet() function is used to get a message from a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is a pointer to the priority of the message, and the fourth parameter is the timeout value.

If the osMessageQueueGet() function succeeds, the Recieve\_Task function then publishes the temperature and humidity readings to MQTT. The sprintf() function is used to format a string, and the HAL\_UART\_Transmit() function is used to transmit a string over UART.

The Recieve\_Task function then starts the timer again.

#include "main.h" #include "cmsis\_os.h" #include"stdio.h" #include"string.h"

**12.Project code :**

#define AHT25\_ADDR 0x70

#define AHT25\_MEASURE\_CMD 0xAC

float temperature, humidity;

I2C\_HandleTypeDef hi2c1;

UART\_HandleTypeDef huart1; UART\_HandleTypeDef huart2; TIM\_HandleTypeDef htim2;

/\* Definitions for SendTask \*/ osThreadId\_t SendTaskHandle;

const osThreadAttr\_t SendTask\_attributes = {

.name = "SendTask",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) *osPriorityNormal*,

};

/\* Definitions for RecieveTask \*/ osThreadId\_t RecieveTaskHandle;

const osThreadAttr\_t RecieveTask\_attributes = {

.name = "RecieveTask",

.stack\_size = 128 \* 4,

.priority = (osPriority\_t) *osPriorityLow*,

};

/\* Definitions for myQueueTemp \*/ osMessageQueueId\_t myQueueTempHandle;

const osMessageQueueAttr\_t myQueueTemp\_attributes = {

.name = "myQueueTemp"

};

/\* Definitions for myTimer \*/ osTimerId\_t myTimerHandle;

const osTimerAttr\_t myTimer\_attributes = {

.name = "myTimer"

};

/\* USER CODE BEGIN PV \*/

typedef struct

{

float temp;

float humi;

}data;

/\* USER CODE END PV \*/

int p=60000;

typedef enum {

*STATE\_INIT*, *STATE\_READ\_SENSOR*, *STATE\_PUBLISH\_MQTT*,

} state\_t;

state\_t state = *STATE\_INIT*;

/\* Private function prototypes \*/

void SystemClock\_Config(void); static void MX\_GPIO\_Init(void);

static void MX\_USART2\_UART\_Init(void); static void MX\_TIM2\_Init(void);

static void MX\_I2C1\_Init(void);

static void MX\_USART1\_UART\_Init(void);

void Send\_Task(void \*argument); void Recieve\_Task(void \*argument); void Callback(void \*argument);

**Project code**

void WE10\_Init (void);

void MQTT\_Init(void);

/\* USER CODE BEGIN PFP \*/

void read\_sensor\_values(float \*temperature, float \*humidity)

{

uint8\_t data[6];

uint8\_t cmd = AHT25\_MEASURE\_CMD;

HAL\_I2C\_Master\_Transmit(&hi2c1, AHT25\_ADDR,&cmd, 1, HAL\_MAX\_DELAY); HAL\_Delay(100);

HAL\_I2C\_Master\_Receive(&hi2c1,0x71, data, 6, HAL\_MAX\_DELAY);

\*humidity = ((float)((data[1] << 12) | (data[2] << 4) | (data[3] >> 4))) / 1048576.0 \* 100.0;

\*temperature = ((float)(((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5])) / 1048576.0 \* 200.0 - 50.0;

}

int main(void)

{

HAL\_Init();

SystemClock\_Config(); MX\_GPIO\_Init(); MX\_USART2\_UART\_Init();

MX\_I2C1\_Init(); MX\_TIM2\_Init(); MX\_USART1\_UART\_Init();

WE10\_Init(); MQTT\_Init(); osKernelInitialize();

myTimerHandle = osTimerNew(Callback, *osTimerPeriodic*, NULL,

&myTimer\_attributes);

myQueueTempHandle = osMessageQueueNew (256, sizeof(uint16\_t), &myQueueTemp\_attributes);

SendTaskHandle = osThreadNew(Send\_Task, NULL, &SendTask\_attributes);

RecieveTaskHandle = osThreadNew(Recieve\_Task, NULL, &RecieveTask\_attributes); osKernelStart();

while (1)

{

}

}

void StateMachine(void \*argument)

{

while(1)

{

switch (state) {

case *STATE\_INIT*:

// Perform initialization tasks if necessary WE10\_Init();

MQTT\_Init();

// Transition to the next state

state=*STATE\_READ\_SENSOR*;

break;

case *STATE\_READ\_SENSOR*:

**Project code**

{

read\_sensor\_values(&temperature, &humidity);

state=*STATE\_PUBLISH\_MQTT*;

}

break;

case *STATE\_PUBLISH\_MQTT*:

{

// Receive data from external source

Send\_Task(&argument);

Recieve\_Task(&argument); Callback(&p);

// Transition to the next state

state=*STATE\_READ\_SENSOR*;

}

break;

}

}

}

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_SlaveConfigTypeDef sSlaveConfig = {0}; TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/ htim2.Instance = TIM2; htim2.Init.Prescaler = 15999;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP; htim2.Init.Period = 60000;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1; htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE; if (HAL\_TIM\_Base\_Init(&htim2) != *HAL\_OK*)

{

Error\_Handler();

}

sSlaveConfig.SlaveMode = TIM\_SLAVEMODE\_EXTERNAL1; sSlaveConfig.InputTrigger = TIM\_TS\_ITR0;

if (HAL\_TIM\_SlaveConfigSynchro(&htim2, &sSlaveConfig) != *HAL\_OK*)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET; sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE; if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) !=

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

}

void SystemClock\_Config(void)

*HAL\_OK* )

{

**Project code**

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0}; RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

HAL\_RCC\_PWR\_CLK\_ENABLE();

HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI; RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON; RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT; RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != *HAL\_OK*)

{

Error\_Handler();

}

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI; RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1; RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1; RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != *HAL\_OK*)

{

Error\_Handler();

}

}

static void MX\_I2C1\_Init(void)

{

hi2c1.Instance = I2C1;

hi2c1.Init.ClockSpeed = 100000; hi2c1.Init.DutyCycle = I2C\_DUTYCYCLE\_2; hi2c1.Init.OwnAddress1 = 0;

hi2c1.Init.AddressingMode = I2C\_ADDRESSINGMODE\_7BIT; hi2c1.Init.DualAddressMode = I2C\_DUALADDRESS\_DISABLE; hi2c1.Init.OwnAddress2 = 0;

hi2c1.Init.GeneralCallMode = I2C\_GENERALCALL\_DISABLE; hi2c1.Init.NoStretchMode = I2C\_NOSTRETCH\_DISABLE;

if (HAL\_I2C\_Init(&hi2c1) != *HAL\_OK*)

{

Error\_Handler();

}

}

static void MX\_USART1\_UART\_Init(void)

{

huart1.Instance = USART1; huart1.Init.BaudRate = 115200;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1; huart1.Init.Parity = UART\_PARITY\_NONE; huart1.Init.Mode = UART\_MODE\_TX\_RX; huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE; huart1.Init.OverSampling = UART\_OVERSAMPLING\_16; if (HAL\_UART\_Init(&huart1) != *HAL\_OK*)

{

Error\_Handler();

}

}

static void MX\_USART2\_UART\_Init(void)

{

huart2.Instance = USART2; huart2.Init.BaudRate = 115200;

**Project code**

huart2.Init.WordLength = UART\_WORDLENGTH\_8B; huart2.Init.StopBits = UART\_STOPBITS\_1; huart2.Init.Parity = UART\_PARITY\_NONE; huart2.Init.Mode = UART\_MODE\_TX\_RX; huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE; huart2.Init.OverSampling = UART\_OVERSAMPLING\_16; if (HAL\_UART\_Init(&huart2) != *HAL\_OK*)

{

Error\_Handler();

}

}

static void MX\_GPIO\_Init(void)

{

HAL\_RCC\_GPIOA\_CLK\_ENABLE();

HAL\_RCC\_GPIOB\_CLK\_ENABLE();

}

void Send\_Task(void \*argument)

{

data DatatoSend;

while (1) {

read\_sensor\_values(&temperature, &humidity); DatatoSend.temp = temperature; DatatoSend.humi = humidity;

osMessageQueuePut(myQueueTempHandle, &DatatoSend, 0, 0);

osDelay(1000); // Delay for 1000 milliseconds (1 second) before the next iteration

}

}

void Recieve\_Task(void \*argument)

{

data retvalue;

for(;;)

{

osMessageQueueGet(myQueueTempHandle,&retvalue,NULL,osWaitForever);

char buffer[128];

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\ n",retvalue.humi);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer,

strlen(buffer),1000);

sprintf (&buffer[0], "CMD+MQTTPUB=base/state/temperature,%.2f\r\ n",retvalue.temp);

HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buffer, strlen(buffer),1000); HAL\_TIM\_Base\_Start\_IT(&htim2);

}

}

void WE10\_Init ()

{

char buffer[128];

/\*\*\*\*\*\*\*\*\* CMD+RESET \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+RESET\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer)); HAL\_Delay(5000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

**Project code**

/\*\*\*\*\*\*\*\*\* CMD+WIFIMODE=1 \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer)); HAL\_Delay(2000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

/\*\*\*\*\*\*\*\*\* CMD+CONTOAP=SSID,PASSWD \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+CONTOAP=MD ARIF,1234567890\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

//memset(&buffer[0],0x00,strlen(buffer)); HAL\_Delay(5000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

/\*\*\*\*\*\*\*\*\* CMD?WIFI\*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD?WIFI\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer)); HAL\_Delay(2000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

}

void MQTT\_Init()

{

char buffer[128]; HAL\_Delay(2000);

/\*\*\*\*\*\*\*\*\*CMD+MQTTNETCFG \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer)); HAL\_Delay(2000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

/\*\*\*\*\*\*\*\*\*CMD+MQTTCONCFG \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-arifm4348-ud8eo8,,,,,,,,,\r\

n"); 

HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer)); HAL\_Delay(2000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

/\*\*\*\*\*\*\*\*\*CMD+MQTTSTART \*\*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

// memset(&buffer[0],0x00,strlen(buffer)); HAL\_Delay(5000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

/\*\*\*\*\*\*\*\*\*CMD+MQTTSUB \*\*\*\*\*\*\*\*\*/

//memset(&buffer[0],0x00,strlen(buffer));

sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n"); HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

**Project code**

HAL\_Delay(2000);

HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buffer, strlen(buffer)); HAL\_UART\_Transmit\_IT(&huart2, (uint8\_t\*)buffer, strlen(buffer));

}

void Callback(void \*argument) {

}

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

if (htim->Instance == TIM5) { HAL\_IncTick();

}

}

void Error\_Handler(void)

{

disable\_irq();

while (1)

{

}

}

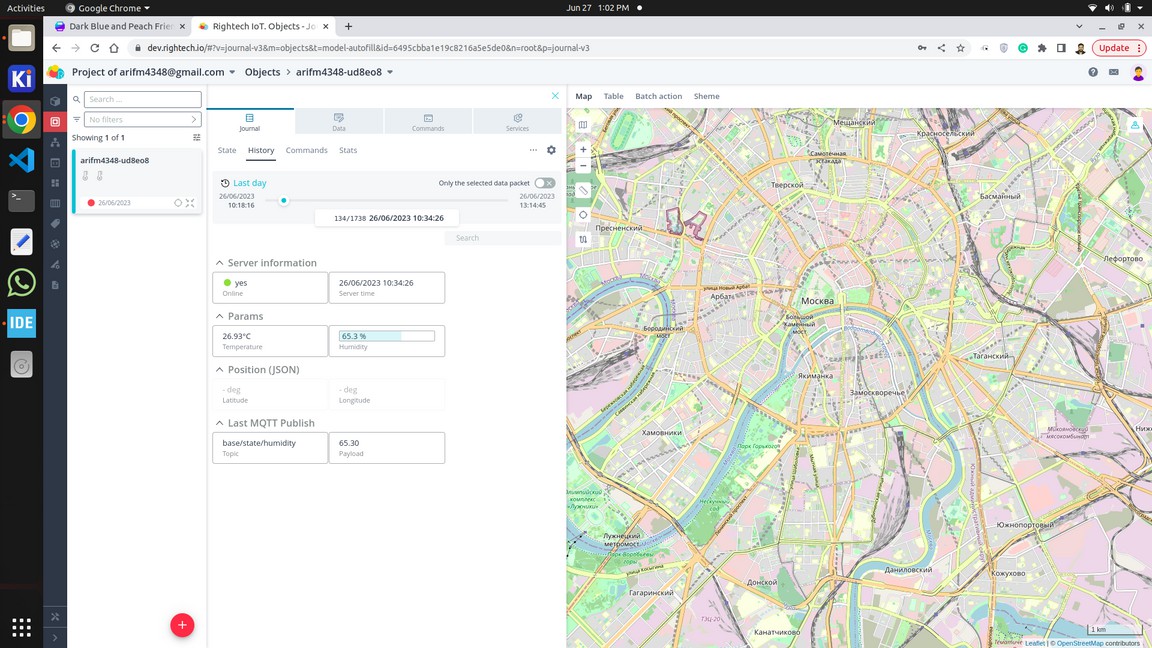
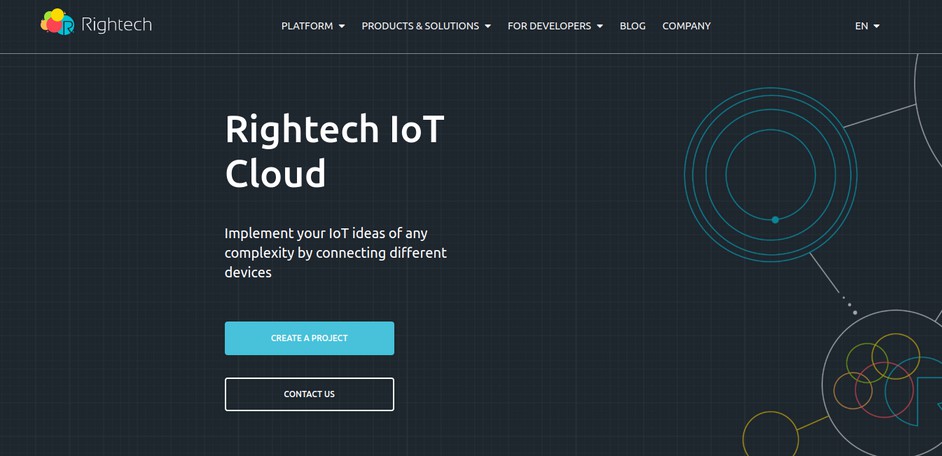
#ifdef USE\_FULL\_ASSERT

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

}

#endif /\* USE\_FULL\_ASSERT \*/

Rightech IoT Cloud is a tool for developers. RIC is independent of specific equipment and protocols, which makes it easier for developers to combine different devices under one solution.Platform tools allow developers to create IoT solutions without extra code and reuse 90% of that solution to launch similar cases.

We can download the stored data from Rightech IoT in different formats and those are

**14.JSON (JavaScript Object Notation):**

 GPX, or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications.

 CSV (comma-separated values) file is a text file that has a specific format that allows data to be saved in a table-structured format.

 JSON (JavaScript Object Notation, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪ

ˌsɒn/) is an open standard file format and data interchange format

that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).

I have downloaded it in JSON format which is given below.



The JSON object you provided contains information about a humidity and temperature reading. The object has the following properties:

 \_org: The organization ID.  \_mid: The message ID.

 \_oid: The object ID.

 humidity: The humidity reading.

 \_ts: The timestamp of the reading.

 id: The ID of the device that sent the reading.  \_lic: The license ID.

 online: Whether the device is online.  time: The time of the reading.

 \_bot: Whether the device is a bot.

 payload: The payload of the message.  \_gid: The group ID.

 topic: The topic of the message.

 temperature: The temperature reading.  \_id: The ID of the JSON object.

The topic property indicates that the message was published to the base/state/humidity topic. The payload property contains the value of the humidity reading, which is 63.93. The temperature property contains the value of the temperature reading, which is 27.77.

{

**15.JSON code :**

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.2,

"\_ts": 1687757972005852,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687757972005,

"\_bot": false, "payload": "64.20",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.26,

"\_ts": 1687757978377336,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687757978377,

"\_bot": false, "payload": "64.26",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.24,

"\_ts": 1687757984860343,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687757984860,

"\_bot": false, "payload": "64.24",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0",

"humidity": 64.22,

**JSON code :**

"\_ts": 1687757990401143,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687757990401,

"\_bot": false, "payload": "64.22",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.42,

"\_ts": 1687757996686076,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687757996686,

"\_bot": false, "payload": "64.42",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.35,

"\_ts": 1687758005919560,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758005919,

"\_bot": false, "payload": "64.35",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.64,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.35,

"\_ts": 1687758008317052,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3",

"online": true,

**JSON code :**

"time": 1687758008317,

"\_bot": false, "payload": "27.73",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/temperature", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.32,

"\_ts": 1687758009699033,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758009699,

"\_bot": false, "payload": "64.32",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.2,

"\_ts": 1687758015169781,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758015169,

"\_bot": false, "payload": "64.20",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.1,

"\_ts": 1687758020818809,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758020818,

"\_bot": false, "payload": "64.10",

"\_gid": "64632961858b98a72bb922ef",

**JSON code :**

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.15,

"\_ts": 1687758027169890,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758027169,

"\_bot": false, "payload": "64.15",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 64.03,

"\_ts": 1687758033550415,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758033550,

"\_bot": false, "payload": "64.03",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

}, {

"\_org": "64632961858b98a72bb922ef",

"\_mid": "64785fff858b98a72bb94534", "\_oid": "6495cbba1e19c8216a5e5de0", "humidity": 63.99,

"\_ts": 1687758038809783,

"id": "6495cbba1e19c8216a5e5de0", "\_lic": "5d3b5ff00a0a7f30b695afe3", "online": true,

"time": 1687758038809,

"\_bot": false, "payload": "63.99",

"\_gid": "64632961858b98a72bb922ef",

"topic": "base/state/humidity", "temperature": 27.73,

"\_id": "649923cac6849e1137383547"

**PROJECT REPORT**

**11**

# Questions? Contact us.

Mahammad Arif Kotwal [arifm4348@gmail.com](mailto:arifm4348@gmail.com) 8123150453

