PAHO-MQTT User Manual

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2018-04-28	v0.1	Armink	Initial version
2018-05-28	v1.0	Armink	Synchronous Update
2018-06-29	v1.1	SummerGift	Add UM Document

Table of contents

Chapter 1

MQTT Software Package Introduction

Paho MQTT It is a client of MQTT protocol implemented by Eclipse. This package is in Eclipse paho-mqtt

A set of MQTT client programs designed based on the source code package.

1.1 File Directory Structure

```
pahomqtt
ÿÿÿÿdocs
ÿ ÿÿÿÿfigures ÿ ÿ api.md                    // Document using images
ÿ ÿ introduction.md ÿ ÿ                    // API usage instructions
principle.md ÿ ÿ README.md ÿ ÿ             // Introduction document
samples.md ÿ ÿ user-guide.md ÿ             // Implementation principle
ÿÿÿÿversion.md                             // Document structure description
ÿÿÿÿMQTTClient-RT                          // Package example
ÿÿÿÿMQTTPacket ÿÿÿÿsamples ÿ                // Instructions
mqtt_sample.c ÿÿÿÿtests ÿ ÿ                // Edition
README.md ÿÿÿÿSConscript                   // Migrate files
                                           // Source file
                                           // Example code
                                           // Software package application sample code
                                           //mqtt functional test program
    LICENSE                                // Package License
                                           // Software package instructions
                                           //RT-Thread default build script
```

1.2 RT-Thread Software Package Features

The RT-Thread MQTT client features are as follows:

• Automatic reconnection after disconnection

The RT-Thread MQTT software package implements a disconnection reconnection mechanism. If the connection is lost due to a network outage or instability, the system will maintain the login status, reconnect, and automatically resubscribe to topics. This improves connection reliability and enhances the usability of the software package.

• Pipe model, non-blocking API

It reduces programming difficulty, improves code running efficiency, and is suitable for situations with high concurrency and small data volumes.

• Event callback mechanism

Custom callback functions can be executed when a connection is established, a message is received, or a connection is disconnected.

• TLS encrypted transmission

MQTT can be transmitted using TLS encryption to ensure data security and integrity.

1.3 MQTT Overview

MQTT (Message Queuing Telemetry Transport) is a lightweight communication protocol based on the publish/subscribe model. Built on TCP/IP, it was released by IBM in 1999. MQTT's greatest advantage is that it can provide real-time, reliable messaging services for connecting remote devices with minimal code and limited bandwidth. As a low-overhead, low-bandwidth instant messaging protocol, it has widespread application in the Internet of Things, small devices, and mobile applications.

MQTT is a client-server publish/subscribe messaging transport protocol. Its lightweight, simple, open, and easy-to-implement characteristics make it widely applicable in many scenarios, including constrained environments such as machine-to-machine (M2M) communications and the Internet of Things (IoT). It is widely used in sensors communicating via satellite links, medical devices with occasional dial-up connections, smart homes, and other small devices.

1.4 MQTT Function Introduction

The MQTT protocol runs on top of TCP/IP or other network protocols. It establishes a connection between a client and a server, providing ordered, lossless, bidirectional byte-stream transmission between the two. When application data is sent over the MQTT network, MQTT associates the associated Quality of Service (QoS) with the topic name. Its features include:

1. Uses the publish/subscribe messaging model, which provides one-to-many message distribution to achieve decoupling from applications. 2. Provides a message transmission mechanism that shields payload content.

3. Provides three qualities of service (QoS) for transmitted messages:

1. At most once. Message loss or duplication may occur at this level, and message publishing depends on the underlying TCP/IP network. That is: <=1
2. At most once: This level ensures that the message arrives, but the message may be duplicated. That is: >= 1 3. Exactly once: This level ensures that the message arrives only once. That is: = 1. In some billing systems with strict requirements This level can be used.

4. Minimize data transmission and protocol exchange (the protocol header is only 2 bytes) to reduce network traffic. 5. Notification mechanism to notify both parties of abnormal interruption.

1.4.1 MQTT Client

An application or device using the MQTT protocol always establishes a network connection to the server. The client can:

- Establish a connection with the server • Publish information that other clients may subscribe to
- Receive messages published by other clients • Unsubscribe from subscribed messages

1.4.2 MQTT Server

The MQTT server is called a "message broker" (Broker), which can be an application or a device. Between message publishers and subscribers, it can:

- Accepts network connections from clients • Receives application information published by clients • Handles subscription and unsubscription requests from clients • Forwards application messages to subscribed clients

1.4.3 Methods in the MQTT Protocol

The MQTT protocol defines methods (also known as actions) that represent operations on specific resources. These resources can represent pre-existing data or dynamically generated data, depending on the server implementation. Typically, a resource refers to a file or output on the server.

- Connect: Waits for a connection to be established with the server. • Disconnect: Waits for the MQTT client to complete its work and disconnect the TCP/IP session with the server. • Subscribe: Waits for a subscription to be completed. • UnSubscribe: Waits for the server to unsubscribe the client from one or more topics. • Publish: The MQTT client sends a message request and returns to the application thread after the message is sent.

1.4.4 Subscriptions, Topics, and Sessions in the MQTT Protocol

- Subscription

  Subscription contains topic filter and maximum quality of service (QoS). Subscription is associated with a session. (Session) association. A session can contain multiple subscriptions. Each subscription in a session has a different topic filter.

- Session

  Each time a client establishes a connection with a server, a session occurs. This stateful interaction exists between the client and server. A session exists within a single network, but may also span multiple consecutive network connections between the client and server.

- Topic Name

  A tag connected to an application message that matches a server's subscription. The server sends the message to every client that subscribed to the matching tag.

- Topic Filter

  A wildcard filter for topic names, used in a subscription expression, to indicate that the subscription matches multiple topics.

- Payload

  The specific content received by the message subscriber.

- Application Message

  The MQTT protocol transmits application data over the network. When application messages are transmitted via MQTT, they have associated quality of service (QoS) and topics.

- MQTT Control Packet

  A packet of information sent over a network connection. The MQTT specification defines fourteen different types of control messages, one of which (the PUBLISH message) is used to transmit application messages.

Chapter 2

MQTT sample program

2.1 Example Code Explanation

The following explains the MQTT sample code provided by RT-Thread. The test server uses the Eclipse test server.

The address is iot.eclipse.org , port 1883, and the sample code for the MQTT function is as follows:

```
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#include <rtthread.h>

#define DBG_ENABLE
#define DBG_SECTION_NAME        "[MQTT] "
#define DBG_LEVEL               DBG_LOG
#define DBG_COLOR
#include <rtdbg.h>

#include "paho_mqtt.h"
#define MQTT_URI                "tcp://iot.eclipse.org:1883" // Configuration test
    server address
#define MQTT_USERNAME           "admin"
#define MQTT_PASSWORD           "admin"
#define MQTT_SUBTOPIC           "/mqtt/test" "/          // Set the subscription topic
#define MQTT_PUBTOPIC           mqtt/test"               // Set the push topic
#define MQTT_WILLMSG            "Goodbye!"               // Set the last message

/* Define the MQTT client environment structure*/
static MQTTClient client;
```

```
/* MQTT subscription event custom callback
function*/ static void mqtt_sub_callback(MQTTClient *c, MessageData *msg_data) {

    *((char *)msg_data->message->payload + msg_data->message->payloadlen)
            = '\0';
    LOG_D("mqtt sub callback: %.*s %.*s", msg_data-
                >topicName->lenstring.len, msg_data->topicName-
                >lenstring.data, msg_data->message->payloadlen,
                (char *)msg_data->message->payload);


    return;
}

/* MQTT subscription event default callback
function*/ static void mqtt_sub_default_callback(MQTTClient *c, MessageData
    msg_data)
{
    *((char *)msg_data->message->payload + msg_data->message->payloadlen)
            = '\0';
    LOG_D("mqtt sub default callback: %.*s %.*s", msg_data->topicName-
                >lenstring.len, msg_data->topicName->lenstring.data,
                msg_data->message->payloadlen, (char *)msg_data-
                >message->payload);

    return;
}

/* MQTT connection event callback
function*/ static void mqtt_connect_callback(MQTTClient *c) {

    LOG_D("inter mqtt_connect_callback!");
}

/* MQTT online event callback function*/
static void mqtt_online_callback(MQTTClient *c) {

    LOG_D("inter mqtt_online_callback!");
}

/* MQTT offline event callback function*/
static void mqtt_offline_callback(MQTTClient *c) {

    LOG_D("inter mqtt_offline_callback!");
}
```

```
/**
 * This function creates and configures the MQTT client.
 *
 * @param void
 *
 * @return none
 */
static void mq_start(void) {

    /* Use MQTTPacket_connectData_initializer to initialize the condata parameter*/
    MQTTPacket_connectData condata = MQTTPacket_connectData_initializer; static char cid[20] = { 0 };


    static int is_started = 0; if (is_started)
    {

        return;

    } /* Configure MQTT structure content parameters*/
    {
        client.uri = MQTT_URI;

        /* Generate a random client ID */
        rt_snprintf(cid, sizeof(cid), "rtthread%d", rt_tick_get());

        /* ÿ ÿ ÿ ÿ ÿ ÿ */
        memcpy(&client.condata, &condata, sizeof(condata)); client.condata.clientID.cstring
        = cid; client.condata.keepAliveInterval = 60;
        client.condata.cleansession = 1;

        client.condata.username.cstring = MQTT_USERNAME;
        client.condata.password.cstring = MQTT_PASSWORD;

        /* Configure MQTT last words parameters*/
        client.condata.willFlag = 1;
        client.condata.will.qos = 1;
        client.condata.will.retained = 0;
        client.condata.will.topicName.cstring = MQTT_PUBTOPIC;
        client.condata.will.message.cstring = MQTT_WILLMSG;

        /* Allocate buffer */
        client.buf_size = client.readbuf_size = 1024; client.buf =
        malloc(client.buf_size); client.readbuf =
        malloc(client.readbuf_size); if (!(client.buf && client.readbuf)) {
```

```
                    LOG_E("no memory for MQTT client buffer!"); goto _exit;

            }

            /* Set event callback function*/
            client.connect_callback = mqtt_connect_callback; client.online_callback =
            mqtt_online_callback; client.offline_callback = mqtt_offline_callback;


            /* Set up subscription table and event callback function*/
            client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC;
            client.messageHandlers[0].callback = mqtt_sub_callback; client.messageHandlers[0].qos
            = QOS1;

            /* Set the default subscription topic */
            client.defaultMessageHandler = mqtt_sub_default_callback;
        }

        /* Run MQTT client*/
        paho_mqtt_start(&client); is_started = 1;


_exit:
        return;
}

/**
  * This function pushes a message to a specific MQTT topic.
  *
  * @param send_str publish message
  *
  * @return none
  */
static void mq_publish(const char *send_str) {

        MQTTMessage message;
        const char *msg_str = send_str; const char
        *topic = MQTT_PUBTOPIC;
        message.qos = QOS1;                                          //Message level
        message.retained = 0;
        message.payload = (void *)msg_str; message.payloadlen
        = strlen(message.payload);

        MQTTPublish(&client, topic, &message);

        return;
```

```
}

#ifdef RT_USING_FINSH
#include <finsh.h>
FINSH_FUNCTION_EXPORT(mq_start, startup mqtt client);
FINSH_FUNCTION_EXPORT(mq_publish, publish mqtt msg); #ifdef
FINSH_USING_MSH
MSH_CMD_EXPORT(mq_start, startup mqtt client);

int mq_pub(int argc, char **argv) {

        if (argc != 2) {

                rt_kprintf("More than two input parameters err!\n");
                return 0;

        } mq_publish(argv[1]);

        return 0;
}
MSH_CMD_EXPORT(mq_pub, publish mqtt msg); #endif /*
FINSH_USING_MSH */ #endif /*
RT_USING_FINSH */
```

2.2 Running the Example

Running the above functional sample code in msh can subscribe to topics on the server and push messages to specific topics.

The function sample code running effect is as follows:

- Start the MQTT client, connect to the broker server and subscribe to the topic:

```
msh />mq_start inter                    /* Start the MQTT client to connect to the Eclipse server*/
mqtt_connect_callback! /* Server connection is successful, call the connection callback function to print the service
    Device information*/
ipv4 address port: 1883 [MQTT]
HOST = 'iot.eclipse.org' msh />[MQTT] Subscribe
#0 /mqtt/test OK! /* Subscribe to topic /mqtt/test successfully*/ inter mqtt_online_callback! /* MQTT goes online
successfully, call the online callback function*/
msh />
```

- Publish a message to a specified topic as a publisher:

```
msh />mq_pub hello-rtthread /* Send hello-rtthread message to the specified topic*/
msh />mqtt sub callback: /mqtt/test hello-rtthread /* Receive message and execute callback
        function*/
msh />
```

Chapter **3**

How **MQTT** works

3.1 How the MQTT Protocol Works

In the MQTT protocol, there are three roles: publisher, broker (server), and subscriber. The publisher and subscriber of a message are both clients, the message broker is the server, and the message publisher can also be a subscriber. The relationship between these three roles is shown in the following figure:

Figure **3.1:** *MQTT* Working principle diagram

In the actual use of the MQTT protocol, the following process is generally followed:

• The publisher publishes messages to the specified topic through the proxy server. •
The subscriber subscribes to the required topic through the proxy server. • After
the subscription is successful, if a publisher publishes a message to the topic subscribed by the subscriber, the subscriber will receive the proxy message.
The server pushes messages, which enables efficient data exchange.

Chapter 4

MQTT Usage Guide

4.1 Preparation

First, you need to download the MQTT software package and add it to the project. In the BSP directory, use the menucon-fig command to open the env configuration interface, and in RT-Thread online packages ỳ IoT - internet of things

Select the Paho MQTT software package, and the operation interface is as shown below:

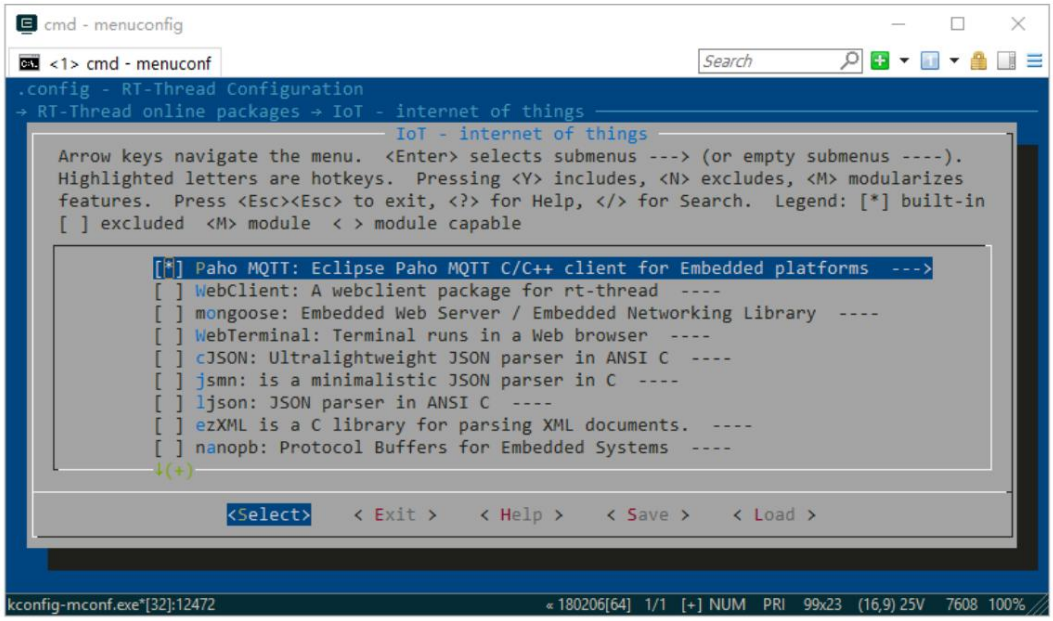

Figure 4.1: Selected *MQTT broadcast* Software Package

Enable functional examples to facilitate testing of MQTT functions:

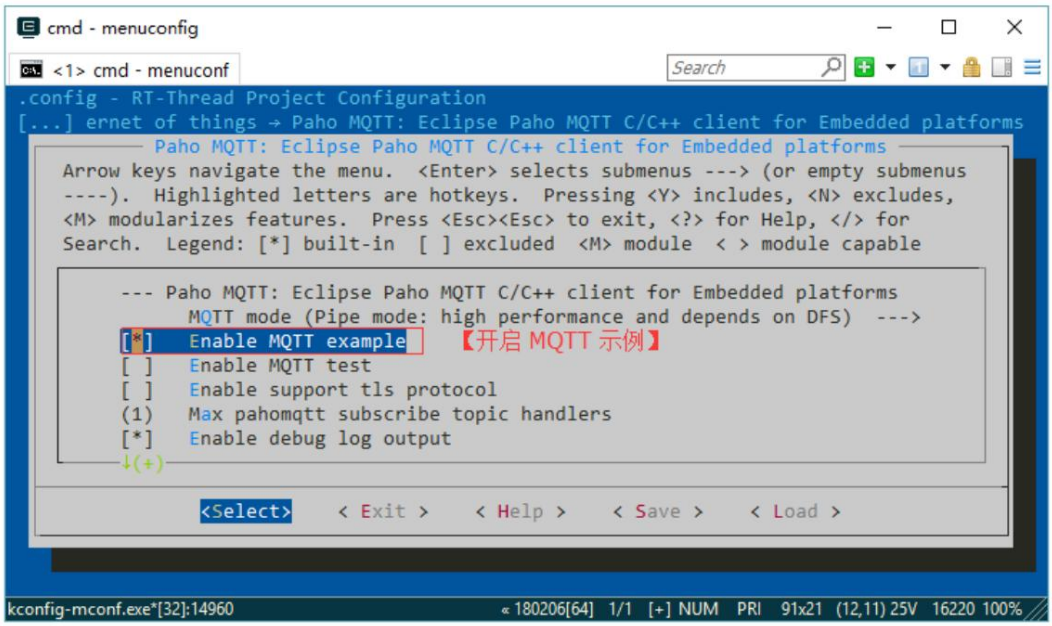

Figure **4.2:** Open *MQTT* Software package testing routines

The configuration items are described as follows:

```
--- Paho MQTT: Eclipse Paho MQTT C/C++ client for Embedded platforms
        MQTT mode (Pipe mode: high performance and depends on DFS) --->#ÿ ÿ
                Function
        [*] Enable MQTT example                          #Enable MQTT function example
        [ ] Enable MQTT test                             #Open MQTT test routine
        [ ] Enable support tls protocol                  # Enable TLS secure transmission option
        (1) Max pahomqtt subscribe topic handlers #Set the maximum number of subscriptions to a topic
        [*] Enable debug log output #Select the software # Enable debug log output
        version (latest) --->                            package version, the default is the latest version
```

After selecting the appropriate configuration items, use the pkgs --update command to download the software package and add it to the project.

4.2 Usage Process

This section introduces the configuration parameters and usage of the MQTT software package.

4.2.1 Setting proxy information

First, you need to set the proxy server address, user name, password and other necessary information. Take the MQTT sample as an example.

The following settings:

```
#define MQTT_URI "tcp://iot.eclipse.org:1883" //Set up service address
#define MQTT_USERNAME "admin" //Proxy service Server Username
#define MQTT_PASSWORD "admin" //Proxy service password
#define MQTT_SUBTOPIC "/mqtt/test" //Subscription Topic
#define MQTT_PUBTOPIC "/mqtt/test" //Push Topic
#define MQTT_WILLMSG "Goodbye!" //Set disconnect notification message
```

4.2.2Configure MQTT client structure

Next, you need to initialize the MQTT package client instance and write the data set in the previous step into the client instance Configuration items, perform necessary configuration on the client. In this step, you need to do the following:

- Set the server address, server account, password and other information. The sample code is as follows:

```
/* Configure connection parameters*/
memcpy(&client.condata, &condata, sizeof(condata));
client.condata.clientID.cstring = cid;
client.condata.keepAliveInterval = 60;
client.condata.cleansession = 1;
client.condata.username.cstring = MQTT_USERNAME; //Set up account
client.condata.password.cstring = MQTT_PASSWORD; //Set password
```

- Set the message level, push topic, and disconnection notification message configurations. The example is as follows:

```
/* Configure disconnect notification message*/
client.condata.willFlag = 1;
client.condata.will.qos = 1;
client.condata.will.retained = 0;
client.condata.will.topicName.cstring = MQTT_PUBTOPIC; client.condata.will.message.cstring = MQTT_WILLMSG; //Set the push topic //Set disconnect notification information
```

- Set the event callback function. Here you need to set the callback function for the event, such as connection success event, online success event, Offline events, etc. The sample code is as follows:

```
/* Set the event callback function. The callback function needs to be written by yourself. An empty function is left for the callback function in the routine.
        */
client.connect_callback = mqtt_connect_callback; client.online_callback =                    //Set the connection callback
mqtt_online_callback; client.offline_callback = mqtt_offline_callback;                       function //Set the online callback
                                                                                             function //Set the offline callback function
```

- Set up the client subscription table. The MQTT client can subscribe to multiple topics at the same time, so you need to maintain a subscription table. In this step, you need to set parameters for each topic subscription, including the topic name, the subscription callback function, and the message level. The code example is as follows:

```
/* Configure subscription
table*/ client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC; //Set the first subscription
          Topic
client.messageHandlers[0].callback = mqtt_sub_callback; // Set the callback for this subscription
          function
client.messageHandlers[0].qos = QOS1;                                                        //Set the subscription message
          grade
/* set default subscribe event callback */ client.defaultMessageHandler =
mqtt_sub_default_callback; //Set a default callback function. If a subscribed Topic does not have a callback function set, the
          default callback function will be used.
```

4.2.3 Start the **MQTT** client

After configuring the MQTT client instance, you need to start the client. The code example is as follows:

```
/* Run the MQTT client */
paho_mqtt_start(&client);
```

After starting the MQTT client, the client will automatically connect to the proxy server and automatically subscribe to the set topic.

Execute the callback function according to the event to process the data.

4.2.4 Pushing messages to a specified **topic**

After successfully connecting to the server, you can push messages to the specified Topic through the proxy server.

To set the message content, topic, message level and other configurations, the sample code is as follows:

```
MQTTMessage message;
const char *msg_str = send_str;
const char *topic = MQTT_PUBTOPIC; message.qos =          //Set the specified Topic
QOS1; message.retained = 0;                               //Set the message level

message.payload = (void *)msg_str; message.payloadlen =   //Set message content
strlen(message.payload);
MQTTPublish(&client, topic, &message);                    //Start pushing messages to the specified Topic
```

breath

4.3 Operation Effect

The demo example can show the functions of connecting to the server, subscribing to a topic, and pushing messages to a specified topic, as shown below:

```
msh />mq_start                                            /* Start the MQTT client to connect to the proxy server
    */
inter mqtt_connect_callback! ipv4 address port:           /* Connection successful, run online callback function*/
1883
[MQTT] HOST = 'iot.eclipse.org'
msh />[MQTT] Subscribe
inter mqtt_online_callback! msh />mq_pub hello-           /* Successfully launched, running online callback function*/
rtthread msh />mqtt sub callback: /mqtt/test hello-       /* Push message to the specified Topic*/
rtthread /* Receive the message and execute the callback
    function*/
```

4.4 Notes

Please note that MQTT_USERNAME and MQTT_PASSWORD should be filled in correctly .

The MQTT_PASSWORD value is incorrect and the MQTT client cannot connect to the MQTT server correctly.

4.5 References

- MQTT official website
- Paho official website
- Introduction to IBM MQTT
- Eclipse paho.mqtt source code

Chapter 5

Introduction to the MQTT API

5.1 Subscription List

Paho MQTT uses the subscription list format to subscribe to multiple topics. The subscription list is stored in

In the MQTTClient structure instance, configure it before starting MQTT as follows:

```
... // Omit code

MQTTClient client;

... // Omit code

/* set subscribe table and event callback */
client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC;
client.messageHandlers[0].callback = mqtt_sub_callback; client.messageHandlers[0].qos
= QOS1;
```

Please refer to the Samples section for detailed code explanation. The maximum number of subscription lists can be set by the menuconfig

Max pahomqtt subscribe topic handlers option is configured.

5.2 callback

paho-mqtt uses callback to provide users with the working status of MQTT and the processing of related events.

It needs to be registered and used in the MQTTClient structure instance.

callback name	describe
connect_callback	MQTT connection success callback

callback name	describe
online_callback	Callback when the MQTT client successfully goes online
offline_callback	Callback when the MQTT client is disconnected
defaultMessageHandler	Default subscription message receiving callback
messageHandlers[x].callback	The corresponding subscription message receiving callback in the subscription list

Users can use the defaultMessageHandler callback to handle received subscription messages by default, or use

The messageHandlers subscription list provides an independent

Subscription message receiving callback.

5.3 MQTT_URI

paho-mqtt provides uri parsing function, which can parse domain name address, ipv4 and ipv6 address, and can parse

For URIs of the tcp:// and ssl:// types, users only need to fill in the available URIs as required.

- Example uri:

```
domain type
tcp://iot.eclipse.org:1883

IPv4 type
tcp://192.168.10.1:1883
ssl://192.168.10.1:1884

IPv6 type
tcp://[fe80::20c:29ff:fe9a:a07e]:1883
ssl://[fe80::20c:29ff:fe9a:a07e]:1884
```

5.4 paho_mqtt_start interface

- Function: Start the MQTT client and subscribe to the corresponding topic according to the configuration items.

- Function prototype:

```
int paho_mqtt_start(MQTTClient *client)
```

- Function parameters:

parameter	describe
client	MQTT client instance object
return	0: Success; Others: Failure

5.5 MQTT Publish Interface

- Function: Publish MQTT messages to the specified Topic.
- Function prototype:

```
int MQTTPublish(MQTTClient *c, const char *topicName, MQTTMessage *
    message)
```

- Function parameters:

parameter	describe
c	MQTT client instance object
topicName	MQTT message publishing topic
message	MQTT message content
return	0: Success; Others: Failure