Realtek MQTT User Guide

This document provides guideline to use MQTT module in SDK.

Table of Contents

1 MQTT Protocol Introduction

MQ Telemetry Transport (MQTT) is a lightweight broker-based publish/subscribe messaging protocol designed to be open, simple, lightweight and easy to implement.

These characteristics make it ideal for use in constrained environments, like low bandwidth or unreliable network and embedded device with limited processor or memory resources.

Features of the protocol include the publish/subscribe message pattern, messaging transport agnostic to the content of the payload, the use of TCP/IP, three qualities of service for message delivery, a small transport overhead, and the Last Will and Testament mechanism.

1.1 Message format

The message header for each MQTT command message contains a fixed header. Some messages also require a variable header and a payload.

Fixed header

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Message Type

The Message types are shown in the table below.

Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)

Mnemonic	Enumeration	Description
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request
PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

1.2 Connect and Keep Alive

CONNECT/ CONNACK/ DISCONNECT

When a TCP/IP socket connection is established from a client to a server, a protocol level session must be created using a CONNECT flow. The client sends a CONNECT message to a server. The server sends a CONNACK message in response to the CONNECT message from the client.

If the server does not receive a CONNECT message within a reasonable amount of time after the TCP/IP connection is established, the server should close the connection.

If the client does not receive a CONNACK message from the server within a reasonable amount of time, the client should close the TCP/IP socket connection, and restart the session by opening a new socket to the server and issuing a CONNECT message.

In both of these scenarios, a "reasonable" amount of time depends on the type of application and the communications infrastructure.

If a client with the same Client ID is already connected to the server, the "older" client must be disconnected by the server before completing the CONNECT flow of the new client.

The DISCONNECT message is sent from the client to the server to indicate that it is about to close its TCP/IP connection. This allows for a clean disconnection, rather than just dropping the line.

Keep Alive timer

The Keep Alive timer is present in the variable header of a MQTT CONNECTmessage.

The Keep Alive timer, measured in seconds, defines the maximum time interval between messages received from a client.

It enables the server to detect that the network connection to a client has dropped, without having to wait for the long TCP/IP timeout.

If the server does not receive a message from the client within one and a half times the Keep Alive time period (the client is allowed "grace" of half a time period), it disconnects the client as if the client had sent a DISCONNECT message.

If a client does not receive a PINGRESP message within a Keep Alive time period after sending a PINGREQ, it should close the TCP/IP socket connection.

The actual value is application-specific, but a typical value is a few minutes. The maximum value is approximately 18 hours. A value of zero (0) means the client is not disconnected.

1.3 Publish

PUBLISH

A PUBLISH message is sent by a client or a server. Each PUBLISH message is associated with a topic name (also known as the Subject or Channel).

PUBACK

A PUBACK message is the response to a PUBLISH message with QoS level 1. A PUBACK message is sent by a server in response to a PUBLISH message from a publishing client, and by a subscriber in response to a PUBLISH message from the server.

When the client receives the PUBACK message, it discards the original message, because it is also received (and logged) by the server.

PUBREC / PUBREL / PUBCOMP

A PUBREC message is the response to a PUBLISH message with QoS level 2.

A PUBREL message is the third message in the QoS 2 protocol flow.

A PUBCOMP message is the fourth and last message in the QoS 2 protocol flow.

1.4 Subscribe

SUBSCRIBE

The SUBSCRIBE message allows a client to register an interest in one or more topic names with the server. Messages published to these topics are delivered from the server to the client as PUBLISH messages. The SUBSCRIBE message also specifies the QoS level at which the subscriber wants to receive published messages.

A server may start sending PUBLISH messages due to the subscription before the client receives the SUBACK message.

A server may choose to grant a lower level of QoS than the client requested.

SUBACK

A SUBACK message is sent by the server to the client to confirm receipt of a SUBSCRIBE message. A SUBACK message contains a list of granted QoS levels. Each level corresponds to a topic name in the corresponding SUBSCRIBE message.

UNSUBSCRIBE

An UNSUBSCRIBE message is sent by the client to the server to unsubscribe from named topics. The client unsubscribes from the list of topics named in the payload.

UNSUBACK

The UNSUBACK message is sent by the server to the client to confirm receipt of an UNSUBSCRIBE message.

1.5 Qos

MQTT delivers messages according to the levels defined in a Quality of Service (QoS). The levels are described below:

QoS level 0: At most once delivery

Client	Message and direction	Server

Client	Message and direction	Server
QoS = 0	PUBLISH ---------->	**Action:** Publish message to subscribers

QoS level 1: At least once delivery

Client	Message and direction	Server
QoS = 0	PUBLISH ---------->	**Action:** Publish message to subscribers

QoS level 2: Exactly once delivery

Client	Message and direction	Server
QoS = 2 DUP = 0 Message ID = x **Action:** Store message	PUBLISH ---------->	**Action:** Store message *or* **Actions:** • Store message ID • Publish message to subscribers
	PUBREC <----------	Message ID = x
Message ID = x	PUBREL ---------->	**Actions:** • Publish message to subscribers • Delete message *or* **Action:** Delete message ID
Action: Discard message	PUBCOMP <----------	Message ID = x

2 MQTT APIs

The MQTT APIs are provided in MQTTClient library for environments of FreeRTOS. It is built on top of MQTTPacket. MQTTPacket is the lowest level library, the simplest and smallest, but hardest to use. It simply deals with serialization and deserialization of MQTT packets.

This sub-section lists the provided APIs for MQTT operations.

2.1 MQTTClientInit

This function triggers to create an MQTT client object.

Syntax

```
void MQTTClientInit(
        MQTTClient* client,
        Network* network,
        unsigned int command_timeout_ms,
        unsigned char* sendbuf,
        size_t sendbuf_size,
        unsigned char* readbuf,
        size_t readbuf_size
);
```

Parameters

client

The client object to use.

network

The network object to use.

command_timeout_ms

Command timeout in milliseconds.

sendbuf

Send buffer.

sendbuf_size

Send buffer size.

readbuf

Read buffer.

readbuf_size

Read buffer size.

Return Value

None.

Remarks

None.

2.2 MQTTConnect

This function triggers to send an MQTT connect packet down the network and wait for a Connack.

Syntax

```
int MQTTConnect (
        MQTTClient* c,
        MQTTPacket_connectData* options
);
```

Parameters

c

The client object to use.

options

Connect options.

Return Value

Return 0 if success, otherwise return -1.

Remarks

None.

2.3 MQTTPublish

This function triggers to send an MQTT publish packet and wait for all acks to complete for all QoSs.

Syntax

```
int MQTTPublish (
        MQTTClient* c,
        const char* topicName,
```

```
        MQTTMessage* message
);
```

Parameters

c

The client object to use.

topicName

The topic to publish to.

message

The message to send.

Return Value

Return 0 if success, otherwise return -1.

Remarks

None.

2.4 MQTTSubscribe

This function triggers to send an MQTT subscribe packet and wait for suback before returning.

Syntax

```
int MQTTSubscribe (
        MQTTClient* c,
        const char* topicFilter,
        enum QoS qos,
        messageHandler messageHandler
);
```

Parameters

c

The client object to use.

topicFilter

The topic filter to subscribe to.

qos

Expected qos of publish message.

messageHandler

Message handler.

Return Value

Return 0 if success, otherwise return -1.

Remarks

None.

2.5 MQTTUnsubscribe

This function triggers to send an MQTT unsubscribe packet and wait for unsuback before returning.

Syntax

```
int MQTTUnsubscribe (
        MQTTClient* c,
        const char* topicFilter
);
```

Parameters

c

The client object to use.

topicFilter

The topic filter to unsubscribe from.

Return Value

Return 0 if success, otherwise return -1.

Remarks

None.

2.6 MQTTDisconnect

This function triggers to send an MQTT disconnect packet and close the connection.

Syntax

```
int MQTTDisconnect (
        MQTTClient* c
);
```

Parameters

c

The client object to use.

Return Value

Return 0 if success, otherwise return -1.

Remarks

None.

3 MQTT Example

3.1 Configuration

An example to use the APIs explained in previous sections is provided in example_mqtt.c. To execute this example automatically when booting, configuration should be set as below.

1) Add patch files to paths:

```
component\common\application\mqtt\MQTTClient\
component\common\application\mqtt\MQTTPacket\
component\common\example\mqtt\
```

2) The CONFIG_EXAMPLE_MQTT in platform_opts.h must be enabled as follows.

```
/* platform_opts.h *./
#define CONFIG_EXAMPLE_MQTT     1
```

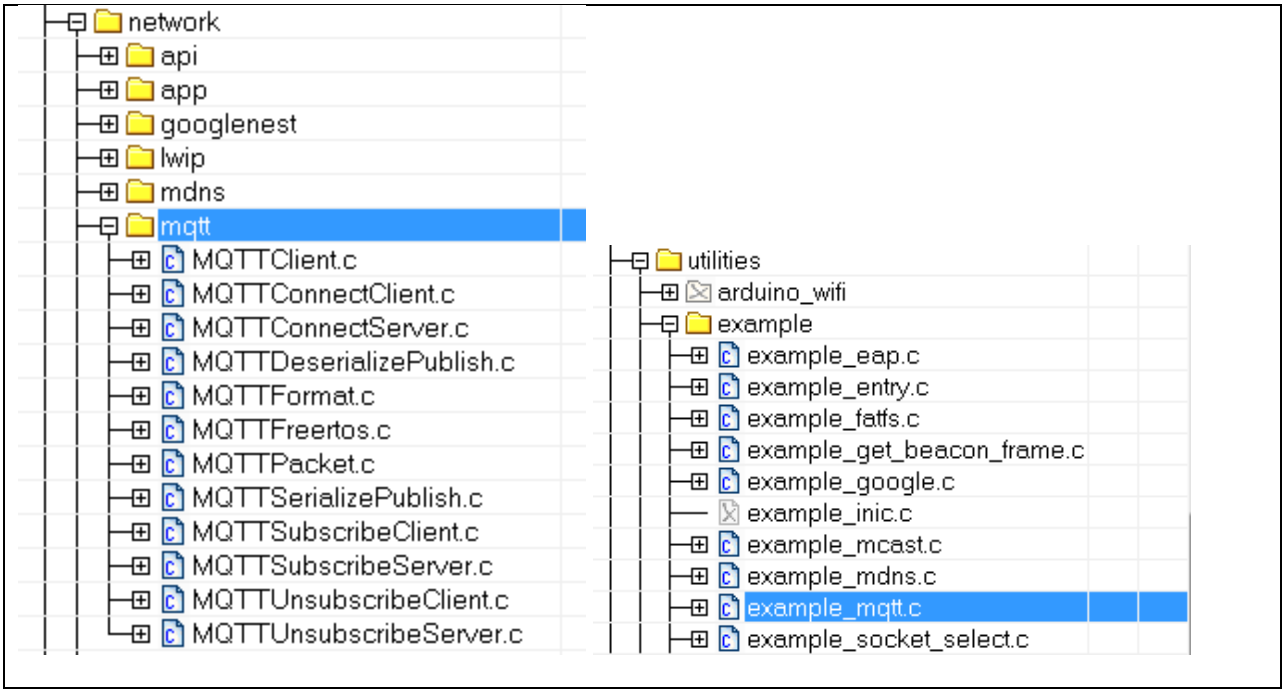
To manage connection exception, LWIP_TCP_KEEPALIVE and LWIP_UART_ADAPTER in lwipopts.h must be enabled as follows.

```
/* lwipopts.h */
#define LWIP_TCP_KEEPALIVE        1
#define LWIP_UART_ADAPTER         1
```

3) Add example_mqtt() to Example_entry.c.

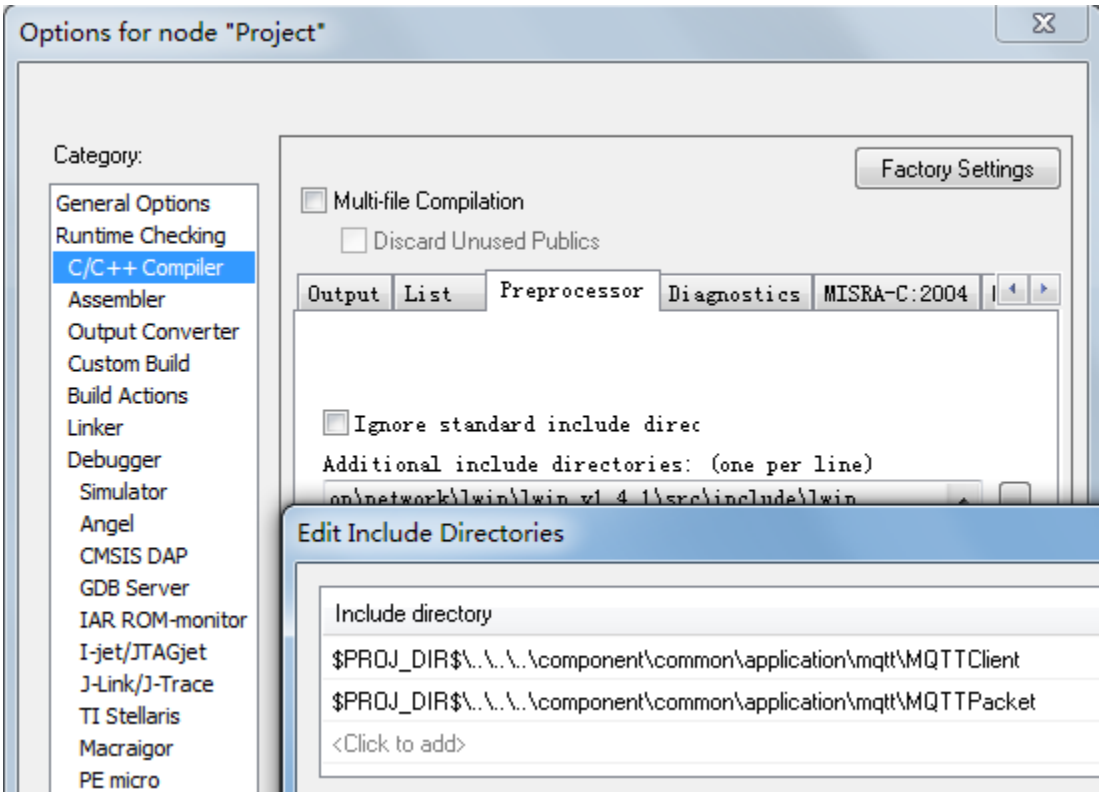
```
#if CONFIG_EXAMPLE_MQTT
#include <mqtt/example_mqtt.h>
#endif

void example_entry(void)
{
#if CONFIG_EXAMPLE_MQTT
        example_mqtt();
#endif
}
```

4) Add MQTT related files to IAR project.



5) Add include directories to project.

```
$PROJ_DIR$\..\..\..\component\common\application\mqtt\MQTTClient
$PROJ_DIR$\..\..\..\component\common\application\mqtt\MQTTPacket
```



3.2 Example Introduction

In the example, MQTT task is created to present MQTT connection and message processing，including:

1) Check WIFI status and wait WIFI to be connected. MQTT will start after device connected with AP and got IP.
2) Establish TCP/IP connection with MQTT server.
3) Send a CONNECT message to server and wait for a CONNACK message from server.
4) Subscribe to a topic, sending SUBSCRIBE to server and wait for SUBACK from server.
5) Publish message to server every 5 seconds.
6) Read and response message. Keep alive with server.
7) If mqtt status is set to MQTT_START, the client will close the TCP/IP socket connection, and restart the session by opening a new socket to the server and issuing a CONNECT message. The client will subscribe to the topic again.

Some strategies are used to manage connection exception.

1) Lwip_select is used to check data arrival and connection exception. Message is read only if tcp data has arrived. If exception fd is set, MQTT will restart.
2) SO_KEEPALIVE and TCP_KEEPIDLE are set to clear TCP buffer when network is bad. For if TCP buffer is full and can't allocate more memory, the situation will last for about 20 minutes until MAX data retries reached, and then MQTT will not restart successfully during this time for allocating memory failed.

3.3 Example Running Log

MQTT running log is shown below:

1) The device should connect WI-FI after initialized. The connected AP must connect to the internet. Use AT command to connect WI-FI or wait for device auto connecting to WIFI If CONFIG_EXAMPLE_WLAN_FAST_CONNECT is set.

```
WIFI initialized

init_thread(51), Available heap 0xc558
[5131]mqtt:Wait Wi-Fi to be connected.

[10142]mqtt:Wait Wi-Fi to be connected.

[15153]mqtt:Wait Wi-Fi to be connected.

[20164]mqtt:Wait Wi-Fi to be connected.
ATW0=TP-LINK_testing
[25175]mqtt:Wait Wi-Fi to be connected.

[ATW0]: _AT_WLAN_SET_SSID_ [TP-LINK_testing]

[MEM] After do cmd, available heap 55752


# ATWC
[ATWC]: _AT_WLAN_JOIN_NET_

Joining BSS by SSID TP-LINK_testing...

RTL8195A[Driver]: set ssid [TP-LINK_testing]

RTL8195A[Driver]: start auth to 8c:21:0a:5a:5a:0e

RTL8195A[Driver]: auth success, start assoc

RTL8195A[Driver]: association success(res=1)

Connected after 1689ms.

Interface 0 IP address : 192.168.1.100

Got IP after 2706ms.


[MEM] After do cmd, available heap 54808


#
[30200]mqtt:MQTT start
```

2) MQTT client starts to connect MQTT server until Wi-Fi is connected and IP address is available. After TCP/IP socket connection is established, a protocol level session must be created using a CONNECT flow. The server sends a CONNACK message in response to a CONNECT message from a client. If not receiving CONNACK in 30s, mqtt status will be set to MQTT_START.

   In the example, the MQTT server address is “gpssensor.ddns.net” and the port is 1883. “gpssensor.ddns.net” is used for test, and customer could build their own MQTT server.

```
[30200]mqtt:MQTT start

[30208]mqtt:Connect Network "gpssensor.ddns.net"

[30331]mqtt:addr = 52.27.59.196

[30694]mqtt:"gpssensor.ddns.net" Connected

[30706]mqtt:Start MQTT connection

[30717]mqtt:Set mqtt status to MQTT_CONNECT

[31058]mqtt:Read packet type is CONNACK

[31069]mqtt:MQTT Connected
```

3) MQTT client subscribes to topic “LASS/Test/Pm25Ameba/#”. After receiving SUBACK, mqtt status is set to MQTT_RUNNING. The client starts to receive PUBLISH message from server.

```
[31078]mqtt:Subscribe to Topic: LASS/Test/Pm25Ameba/#

[31093]mqtt:Set mqtt status to MQTT_SUBTOPIC

[33414]mqtt:Read packet type is SUBACK

[33425]mqtt:grantedQoS: 2

[33433]mqtt:Set mqtt status to MQTT_RUNNING

[33944]mqtt:Read packet type is PUBLISH

[33955]mqtt:Message arrived on topic LASS/Test/Pm25Ameba/FT1_018: one
```

4) If mqtt status is MQTT_RUNNING, the client will publish message every 5s. The published topic is “LASS/Test/Pm25Ameba/FT1_018”. Since the subscribed topic filter is “LASS/Test/Pm25Ameba/#”, the client will receive the message it has published to server.

```
[36975]mqtt:Publish Topic LASS/Test/Pm25Ameba/FT1_018 : 1

[37336]mqtt:Read packet type is PUBACK

[38796]mqtt:Read packet type is PUBLISH

[38807]mqtt:Message arrived on topic LASS/Test/Pm25Ameba/FT1_018: hello from AMEBA 1


[38831]mqtt:send PUBACK

[42839]mqtt:Publish Topic LASS/Test/Pm25Ameba/FT1_018 : 2

[43415]mqtt:Read packet type is PUBACK

[44044]mqtt:Read packet type is PUBLISH

[44055]mqtt:Message arrived on topic LASS/Test/Pm25Ameba/FT1_018: hello from AMEBA 2
```

5) If except_fds is set or read packet type is -1, mqtt status will be set to MQTT_START. except_fds is set for connection closed by server or by client because of KEEPALIVE timeout. Read packet type is -1 for connection closed by server.

```
[2505769]mqtt:except_fds is set

[2505779]mqtt:Set mqtt status to MQTT_START

[2505792]mqtt:MQTT start

[2505799]mqtt:Connect Network "gpssensor.ddns.net"

[2505848]mqtt:addr = 52.27.59.196

[2506370]mqtt:"gpssensor.ddns.net" Connected

[2506383]mqtt:Start MQTT connection

[2506394]mqtt:Set mqtt status to MQTT_CONNECT

[2510063]mqtt:Read packet type is CONNACK

[2510075]mqtt:MQTT Connected

[2510084]mqtt:Subscribe to Topic: LASS/Test/Pm25Ameba/#

[2510100]mqtt:Set mqtt status to MQTT_SUBTOPIC

[2510727]mqtt:Read packet type is SUBACK

[2510739]mqtt:grantedQoS: 2

[2510747]mqtt:Set mqtt status to MQTT_RUNNING

[2511352]mqtt:Read packet type is PUBLISH

[2511364]mqtt:Message arrived on topic LASS/Test/Pm25Ameba/FT1_018: one
```

```
[1068837]mqtt:send PUBACK
[1069845]mqtt:Read packet type is -1
[1069855]mqtt:Set mqtt status to MQTT_START
[1069868]mqtt:MQTT start
[1069876]mqtt:Connect Network "gpssensor.ddns.net"
```