



EC200U&EG91xU Series QuecOpen(SDK) **MQTT Development Guide**

LTE Standard Module Series

Version: 1.1

Date: 2023-10-23

Status: Released



At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our headquarters:

Quectel Wireless Solutions Co., Ltd.

Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China

Tel: +86 21 5108 6236

Email: info@quectel.com

Or our local offices. For more information, please visit:

<http://www.quectel.com/support/sales.htm>.

For technical support, or to report documentation errors, please visit:

<http://www.quectel.com/support/technical.htm>.

Or email us at: support@quectel.com.

Legal Notices

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an “as available” basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

Use and Disclosure Restrictions

License Agreements

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

Copyright

Our and third-party products hereunder may contain copyrighted material. Such copyrighted material shall not be copied, reproduced, distributed, merged, published, translated, or modified without prior written consent. We and the third party have exclusive rights over copyrighted material. No license shall be granted or conveyed under any patents, copyrights, trademarks, or service mark rights. To avoid ambiguities, purchasing in any form cannot be deemed as granting a license other than the normal non-exclusive, royalty-free license to use the material. We reserve the right to take legal action for noncompliance with abovementioned requirements, unauthorized use, or other illegal or malicious use of the material.

Trademarks

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

Third-Party Rights

This document may refer to hardware, software and/or documentation owned by one or more third parties ("third-party materials"). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

Privacy Policy

To implement module functionality, certain device data are uploaded to Quectel's or third-party's servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

Disclaimer

- a) We acknowledge no liability for any injury or damage arising from the reliance upon the information.
- b) We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
- c) While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
- d) We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

Copyright © Quectel Wireless Solutions Co., Ltd. 2023. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2021-11-09	Fei XUE	Creation of the document
1.0	2021-11-12	Fei XUE	First official release
1.1	2023-10-23	Kruskal ZHU	<ol style="list-style-type: none">1. Updated the document name based on the unified QuecOpen naming.2. Added applicable modules: EG912U-GL and EG915U series.3. Updated mqtt_connect_client_info_t structure (Chapter 3.3.2.2)4. Updated mqtt_ssl_config_t structure (Chapter 3.3.2.5)5. Added API function ql_mqtt_client_setopt() (Chapter 3.3.10).

Contents

About the Document	3
Contents	4
Table Index	5
1 Introduction	6
2 Calling Process	7
3 MQTT API	9
3.1. Header File	9
3.2. API Overview	9
3.3. API Description	10
3.3.1. ql_mqtt_client_init	10
3.3.1.1. mqtt_error_code_e	10
3.3.2. ql_mqtt_connect	11
3.3.2.1. mqtt_connection_cb_t	12
3.3.2.2. mqtt_connect_client_info_t	13
3.3.2.3. mqtt_state_exception_cb_t	14
3.3.2.4. mqtt_connection_status_e	14
3.3.2.5. mqtt_ssl_config_t	15
3.3.3. ql_mqtt_publish	17
3.3.3.1. mqtt_request_cb_t	18
3.3.4. ql_mqtt_sub_unsub	18
3.3.5. ql_mqtt_disconnect	19
3.3.5.1. mqtt_disconnect_cb_t	20
3.3.6. ql_mqtt_set_inpub_callback	20
3.3.6.1. mqtt_incoming_publish_cb_t	21
3.3.7. ql_mqtt_client_is_connected	22
3.3.8. ql_mqtt_client_deinit	22
3.3.9. ql_mqtt_onenet_generate_auth_token	22
3.3.10. ql_mqtt_client_setopt	23
3.3.10.1. mqtt_option_e	24
4 Example	25
5 Appendix References	26

Table Index

Table 1: API Overview 9

Table 2: Related Documents 26

Table 3: Terms and Abbreviations..... 26

1 Introduction

Quectel LTE Standard EC200U series and EG91xU family (EG912U-GL and EG915U series) modules support QuecOpen® solution. QuecOpen® is an embedded development platform based on RTOS, which is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see **document [1]**.

This document is applicable to QuecOpen® solution based on CSDK build environment. It introduces the MQTT API, its calling process and example on Quectel EC200U series and EG91xU family modules.

2 Calling Process

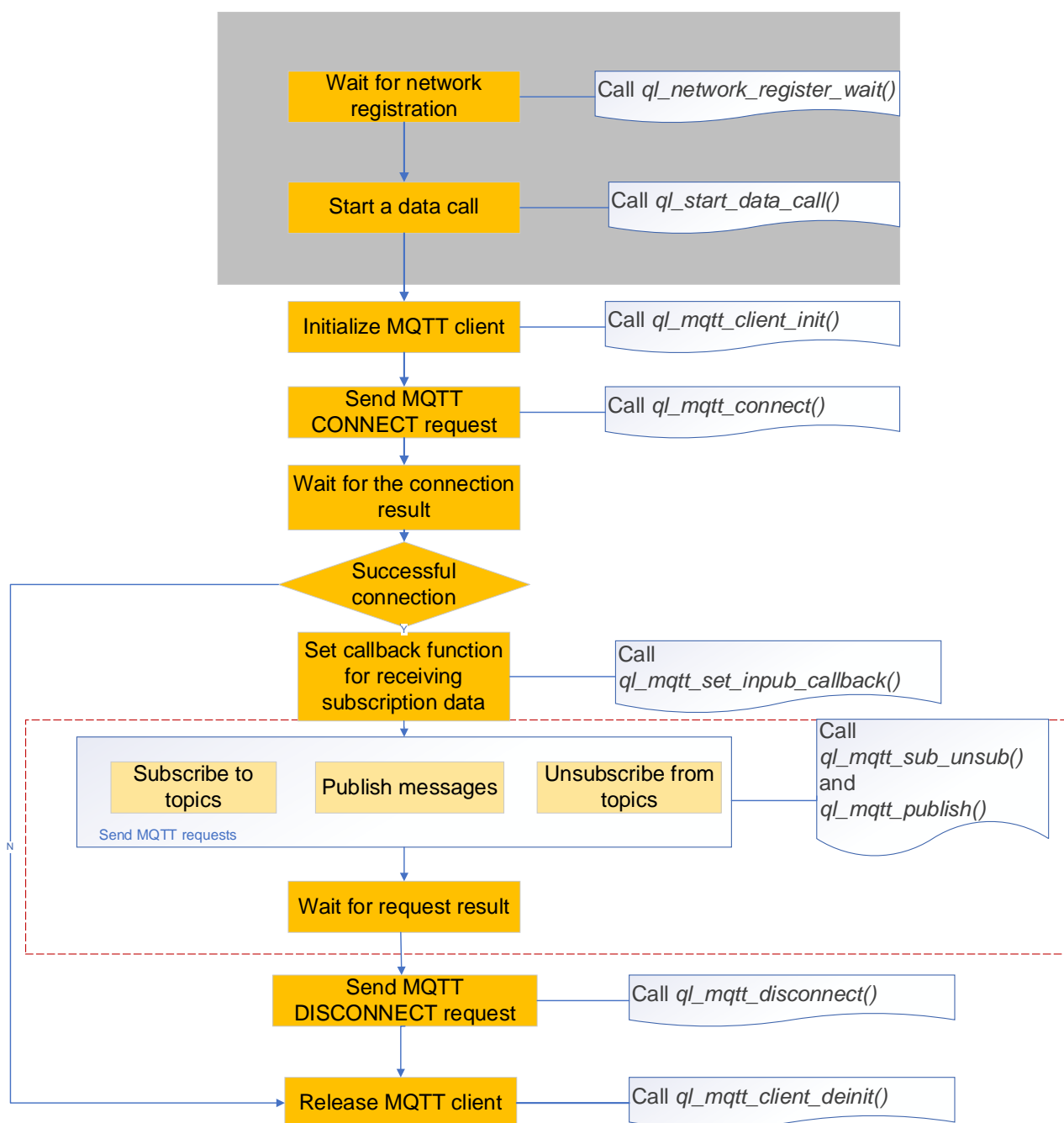


Figure 1: MQTT API Calling Process

After network registration and data call are completed, call MQTT API according to the following steps:

1. Create an MQTT client handle through *ql_mqtt_client_init()*.
2. Send the CONNECT request through *ql_mqtt_connect()* to the server for establishing an MQTT session connection between MQTT client and the server.
3. Set the callback function for receiving messages published by the server through *ql_mqtt_set_inpub_callback()*.
4. Subscribe to topics and unsubscribe from topics through *ql_mqtt_unsub()*.
5. Publish messages of the specified topic through *ql_mqtt_publish()*.
6. Send the DISCONNECT request through *ql_mqtt_disconnect()* to the server for disconnecting the MQTT session between MQTT client and the server.

NOTE

The data transmission via MQTT is based on the specified data channel, so network registration and data call should be completed first to establish the data channel (as shown in grey section of **Figure 1**) before sending MQTT requests. If the data channel has been established in other tasks, network registration and data call can be ignored and *ql_mqtt_client_init()* is directly called to create an MQTT client handle. For more details about data call API, see **document [2]**.

3 MQTT API

3.1. Header File

ql_mqttclient.h, the header file of MQTT API, is in the `\components\ql-kernel\inc` directory. Unless otherwise specified, all the header files mentioned in this document are in this directory

3.2. API Overview

Table 1: API Overview

Function	Description
<i>ql_mqtt_client_init()</i>	Initializes MQTT client and create a new MQTT client handle.
<i>ql_mqtt_connect()</i>	Sends the CONNECT request to the server for establishing an MQTT session connection between the client and the server.
<i>ql_mqtt_publish()</i>	Publishes messages of the specified topic.
<i>ql_mqtt_sub_unsub()</i>	Subscribes to/unsubscribes from the specified topic.
<i>ql_mqtt_disconnect()</i>	Sends the DISCONNECT request to the server for disconnecting the MQTT session between MQTT client and the server.
<i>ql_mqtt_set_inpub_callback()</i>	Sets the callback function for receiving the messages published by the server.
<i>ql_mqtt_client_is_connected()</i>	Queries whether the session has been established between the client and the server.
<i>ql_mqtt_client_deinit()</i>	De-initializes MQTT client.
<i>ql_mqtt_onenet_generate_auth_token()</i>	Generates the password required by OneNET platform.
<i>ql_mqtt_client_setopt</i>	Configures the parameter processing type of the MQTT client handle.

3.3. API Description

3.3.1. ql_mqtt_client_init

This function initializes MQTT client and creates a new MQTT client handle.

- **Prototype**

```
int ql_mqtt_client_init(mqtt_client_t *client, int cid)
```

- **Parameter**

client:

[Out] MQTT client handle.

cid:

[In] Data channel number.

- **Return Value**

See **Chapter 3.3.1.1** for error codes.

3.3.1.1. mqtt_error_code_e

The enumeration of MQTT error codes is defined below:

```
typedef enum{
    MQTTCLIENT_SUCCESS                = 0,
    MQTTCLIENT_INVALID_PARAM          = -1,
    MQTTCLIENT_WOUNDBLOCK             = -2,
    MQTTCLIENT_OUT_OF_MEM             = -3,
    MQTTCLIENT_ALLOC_FAIL             = -4,
    MQTTCLIENT_TCP_CONNECT_FAIL       = -5,
    MQTTCLIENT_NOT_CONNECT            = -6,
    MQTTCLIENT_SEND_PKT_FAIL          = -7,
    MQTTCLIENT_BAD_REQUEST            = -8,
    MQTTCLIENT_TIMEOUT                = -9,
}mqtt_error_code_e
```

- Member

Member	Description
<i>MQTTCLIENT_SUCCESS</i>	Successful execution.
<i>MQTTCLIENT_INVALID_PARAM</i>	Invalid parameter(s).
<i>MQTTCLIENT_WOUNDBLOCK</i>	Operation is blocked, waiting for asynchronous notification of the result.
<i>MQTTCLIENT_OUT_OF_MEM</i>	Out of memory.
<i>MQTTCLIENT_ALLOC_FAIL</i>	Fail to allocate memory.
<i>MQTTCLIENT_TCP_CONNECT_FAIL</i>	Fail to establish TCP connection.
<i>MQTTCLIENT_NOT_CONNECT</i>	No MQTT session connection.
<i>MQTTCLIENT_SEND_PKT_FAIL</i>	Fail to send requests.
<i>MQTTCLIENT_BAD_REQUEST</i>	Bad requests.
<i>MQTTCLIENT_TIMEOUT</i>	Timeout.

3.3.2. ql_mqtt_connect

This function sends the CONNECT request to the server for establishing an MQTT session connection between the client and the server.

- Prototype

```
int ql_mqtt_connect(mqtt_client_t *client, const char *host, mqtt_connection_cb_t cb, void *arg, const struct mqtt_connect_client_info_t *client_info, mqtt_state_exception_cb_t exp_cb)
```

- Parameter

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

host:

[In] MQTT server address, which starts with mqtt:// or mqtt://. For example, mqtt://220.180.239.212:8306 or mqtt://220.180.239.212:8307.

cb:

[In] CONNECT request result callback function. See **Chapter 3.3.2.1** for details.

arg:

[In] Callback argument of CONNECT request result callback function.

client_info:

[In] MQTT client information. See **Chapter 3.3.2.2** for details.

exp_cb:

[In] Callback function for abnormal disconnection of MQTT session. See **Chapter 3.3.2.3** for details.

● Return Value

See **Chapter 3.3.1.1** for error codes.

3.3.2.1. mqtt_connection_cb_t

This callback function is called by kernel to inform the application layer of MQTT CONNECT request results.

● Prototype

```
typedef void (*mqtt_connection_cb_t)(mqtt_client_t *client, void *arg, mqtt_connection_status_e status)
```

● Parameter

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

arg:

[In] Callback argument, which is input by *ql_mqtt_connect()*. See **Chapter 3.3.2** for details.

status:

[In] The result of CONNECT request. See **Chapter 3.3.2.4** for details.

● Return Value

None

3.3.2.2. mqtt_connect_client_info_t

The structure of MQTT client information is defined below:

```
struct mqtt_connect_client_info_t {
    const char *client_id;
    const char* client_user;
    const char* client_pass;
    unsigned short keep_alive;
    unsigned char pkt_timeout;
    unsigned char retry_times;
    const char* will_topic;
    const char* will_msg;
    unsigned char will_qos;
    unsigned char will_retain;
    unsigned char clean_session;
    struct mqtt_ssl_config_t *ssl_cfg;
}
```

● Parameter

Type	Parameter	Description
const char *	<i>client_id</i>	Client ID.
const char *	<i>client_user</i>	The username in CONNECT request. If there is none, set it to <i>NULL</i> .
const char *	<i>client_pass</i>	The password in CONNECT request. If there is none, set it to <i>NULL</i> .
unsigned short	<i>keep_alive</i>	Keep-alive interval between the client and the server. Unit: second.
unsigned char	<i>pkt_timeout</i>	Time for the packet delivery expiration. Unit: second.
unsigned char	<i>retry_times</i>	Retry times when packet delivery expires.
const char *	<i>will_topic</i>	MQTT will topic. If there is none, set it to <i>NULL</i> .
const char*	<i>will_msg</i>	MQTT will message. If there is none, set it to <i>NULL</i> .
unsigned char	<i>will_qos</i>	QoS level of MQTT will message.
unsigned char	<i>will_retain</i>	After a will message is published by the server, if <i>will_retain</i> is <i>TRUE</i> , this message will be reserved by the server. If <i>will_retain</i> is <i>FALSE</i> , this message will be deleted immediately.
unsigned char	<i>clean_session</i>	Enable/disable MQTT session reuse mechanism. 0 Enable

	1	Disable
<code>struct mqtt_ssl_config_t * ssl_cfg</code>		Configuration of MQTT SSL. See Chapter 3.3.2.5 for details. If there is no SSL, set it to <i>NULL</i> .

3.3.2.3. mqtt_state_exception_cb_t

This callback function is called by kernel to inform application layer that the MQTT session is disconnected abnormally. For example, network off-line or keep-alive mechanism timeout leads to disconnection of the MQTT session.

- **Prototype**

```
typedef void(*mqtt_state_exception_cb_t)(mqtt_client_t *client)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

- **Return Value**

None

3.3.2.4. mqtt_connection_status_e

The enumeration of MQTT CONNECT request results is defined below:

```
typedef enum
{
    MQTT_CONNECT_ACCEPTED = 0,
    MQTT_CONNECT_REFUSED_PROTOCOL_VERSION = 1,
    MQTT_CONNECT_REFUSED_IDENTIFIER = 2,
    MQTT_CONNECT_REFUSED_SERVER = 3,
    MQTT_CONNECT_REFUSED_USERNAME_PASS = 4,
    MQTT_CONNECT_REFUSED_NOT_AUTHORIZED = 5,
    MQTT_CONNECT_TCP_CONNECTED_FAILURE = 254,
    MQTT_CONNECT_OUT_OF_MEMORY = 255,
    MQTT_CONNECT_DISCONNECTED = 256,
    MQTT_CONNECT_TIMEOUT = 257
} mqtt_connection_status_e
```

- Member

Member	Description
<i>MQTT_CONNECT_ACCEPTED</i>	Successful connection of MQTT session.
<i>MQTT_CONNECT_REFUSED_PROTOCOL_VERSION</i>	Connection fails due to incorrect protocol version.
<i>MQTT_CONNECT_REFUSED_IDENTIFIER</i>	Connection fails due to the failure of identification of client ID.
<i>MQTT_CONNECT_REFUSED_SERVER</i>	Connection fails due to rejection from the server.
<i>MQTT_CONNECT_REFUSED_USERNAME_PASS</i>	Connection fails due to incorrect username/password.
<i>MQTT_CONNECT_REFUSED_NOT_AUTHORIZED</i>	Connection fails due to the failure of authentication.
<i>MQTT_CONNECT_TCP_CONNECTED_FAILURE</i>	Connection fails due to the failure of TCP connection establishment.
<i>MQTT_CONNECT_OUT_OF_MEMORY</i>	Connection fails due to insufficient memory.
<i>MQTT_CONNECT_DISCONNECTED</i>	Connection fails due to disconnection.
<i>MQTT_CONNECT_TIMEOUT</i>	Connection fails due to timeout.

3.3.2.5. mqtt_ssl_config_t

The structure of SSL configurations in MQTT is defined below:

```

struct mqtt_ssl_config_t{
    int    ssl_ctx_id;
    int    verify_level;
    char   *cacert_path;
    char   *client_cert_path;
    char   *client_key_path;
    char   *client_key_pwd;
    int    ssl_version;
    int    sni_enable;
    int    ssl_negotiate_timeout;
    int    ignore_invalid_certsig;
    int    ignore_multi_certchain_verify;
    uint32_t ignore_certitem
    char   *cacert_buffer;
    bool   client_cert_type;
}

```


- Parameter

Type	Parameter	Description
int	<i>ssl_ctx_id</i>	SSL context ID.
int	<i>verify_level</i>	SSL verification level.
char *	<i>cacert_path</i>	Path of CA certificate.
char *	<i>client_cert_path</i>	Path to client certificate or direct storage of the client certificate content in buffer.
char *	<i>client_key_path</i>	Path to client private key document or direct storage of the client private key document content in buffer.
char *	<i>client_key_pwd</i>	Encrypted password of client private key document.
int	<i>ssl_version</i>	SSL version.
int	<i>sni_enable</i>	Indicates whether to disable/enable Server Name Indication. 0 Disable 1 Enable
int	<i>ssl_negotiate_timeout</i>	Maximum timeout used in SSL negotiation. Unit: second. Range: 10–300.
Int	<i>ignore_invalid_certsign</i>	Indicates whether or not to ignore the invalid certificate signing. 0 Disable 1 Enable
int	<i>ignore_multi_certchain_verify</i>	Indicates whether or not to ignore the multiple level certificate chain verification. 0 Disable 1 Enable
uint32_t	<i>ignore_certitem</i>	Indicates whether the client ignores one or more checks specified in the certificate sent by the server. The parameter applies an accumulative value if the client ignores more than one check. 0 Disable 1 Enable
char *	<i>cacert_buffer</i>	Direct storage of CA certificate content in buffer.
bool	<i>client_cert_type</i>	Indicates whether to store the certificate content directly in buffer. 0 No, in file mode 1 Yes, in buff mode

3.3.3. ql_mqtt_publish

This function publishes messages of the specified topic.

● Prototype

```
int ql_mqtt_publish(mqtt_client_t *client, const char *topic, const void *payload, unsigned short
payload_length, unsigned char qos, unsigned char retain, mqtt_request_cb_t cb, void *arg)
```

● Parameter

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

topic:

[In] Topic of published message.

payload:

[In] Published message.

payload_length:

[In] Length of the published message. Unit: byte.

qos:

[In] QoS level of published message.

retain:

[In] Indicates whether to reserve the published message or not.

- 1 The message is stored on the server after being published
- 0 The message is deleted immediately after being published

cb:

[In] Callback function of PUBLISH request result. See **Chapter 3.3.3.1** for details.

arg:

[In] Callback argument of PUBLISH request result callback function.

● Return Value

See **Chapter 3.3.1.1** for error codes.

3.3.3.1. mqtt_request_cb_t

This callback function is called by kernel to inform the application layer of MQTT subscription, un-subscription and PUBLISH request results.

- **Prototype**

```
typedef void (*mqtt_request_cb_t)(mqtt_client_t *client, void *arg,int err)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

arg:

[In] Callback argument, which is input by *ql_mqtt_sub_unsub()* and *ql_mqtt_publish()*. See **Chapter 3.3.4** and **Chapter 3.3.3** for details.

err:

[In] Request result. See **Chapter 3.3.1.1** for details.

- **Return Value**

None

3.3.4. ql_mqtt_sub_unsub

This function subscribes to or unsubscribes from the specified topic.

- **Prototype**

```
int ql_mqtt_sub_unsub(mqtt_client_t *client, const char *topic, unsigned char qos, mqtt_request_cb_t cb, void *arg,unsigned char sub)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

topic:

[In] Topic to be subscribed to or unsubscribed from.

qos:

[In] QoS level of the topic to be subscribed to or unsubscribed from.

cb:

[In] Callback function of the subscription/un-subscription result. See **Chapter 3.3.3.1** for details.

arg:

[In] Callback argument of the subscription/un-subscription result callback function

sub:

[In] Subscription or un-subscription.

- 1 Subscription
- 0 Un-subscription

● Return Value

See **Chapter 3.3.1.1** for error codes.

3.3.5. ql_mqtt_disconnect

This function sends the DISCONNECT request to the server for disconnecting MQTT session between MQTT client and the server.

● Prototype

```
int ql_mqtt_disconnect(mqtt_client_t *client, mqtt_disconnect_cb_t cb, void *arg)
```

● Parameter

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

cb:

[In] DISCONNECT request result callback function. See **Chapter 3.3.5.1** for details.

arg:

[In] Callback argument of DISCONNECT request result callback function.

● Return Value

See **Chapter 3.3.1.1** for error codes.

3.3.5.1. mqtt_disconnect_cb_t

This callback function is called by kernel to inform application layer of MQTT DISCONNECT request results.

- **Prototype**

```
typedef void (*mqtt_disconnect_cb_t)(mqtt_client_t *client, void *arg, int err)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

arg:

[In] Callback argument, which is input by *ql_mqtt_disconnect()*. See **Chapter 3.3.2** for details.

err:

[In] Request result. See **Chapter 3.3.1.1** for details.

- **Return Value**

None

3.3.6. ql_mqtt_set_inpub_callback

This function sets the callback function for receiving messages published by the server.

- **Prototype**

```
int ql_mqtt_set_inpub_callback(mqtt_client_t *client, mqtt_incoming_publish_cb_t inpub_cb, void *arg)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

inpub_cb:

[In] Callback function for receiving messages published by the server. See **Chapter 3.3.6.1** for details.

arg:

[In] Argument of callback function for receiving messages published by the server.

- **Return Value**

See **Chapter 3.3.1.1** for error codes.

3.3.6.1. mqtt_incoming_publish_cb_t

This callback function is called by kernel to inform application layer that the published messages has been received.

- **Prototype**

```
typedef void (*mqtt_incoming_publish_cb_t)(mqtt_client_t *client, void *arg, int pkt_id, const char *topic,
const unsigned char *payload, unsigned short payload_len)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

arg:

[In] Callback argument, which is input by *ql_mqtt_set_inpub_callback()*. See **Chapter 3.3.6** for details.

pkt_id:

[In] ID of the published message.

topic:

[In] Topic associated with the published message.

payload:

[In] Message body of the published message.

payload_length:

[In] Length of the message body of the published message. Unit: byte.

- **Return Value**

None

3.3.7. ql_mqtt_client_is_connected

This function queries whether the session has been established between the client and the server.

- **Prototype**

```
int ql_mqtt_client_is_connected(mqtt_client_t *client);
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

- **Return Value**

- 1 MQTT session is established.
- 0 MQTT session is not established.

3.3.8. ql_mqtt_client_deinit

This function de-initializes MQTT client.

- **Prototype**

```
int ql_mqtt_client_deinit(mqtt_client_t *client)
```

- **Parameter**

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

- **Return Value**

See **Chapter 3.3.1.1** for error codes.

3.3.9. ql_mqtt_onenet_generate_auth_token

This function generates the password required by OneNET platform.

- **Prototype**

```
char *ql_mqtt_onenet_generate_auth_token(signed long long expire_time,char *product_id,char *device_name,char *version,char *access_key)
```

● Parameter

expire_time:

[In] Expiration time of the token on the OneNET platform. Unit: second.

product_id:

[In] Product ID of OneNET platform.

device_name:

[In] Device name of OneNET platform.

version:

[In] Version of OneNET platform.

access_key:

[In] Device password of OneNET platform.

● Return Value

NULL Failed execution.

Other values Successful execution. After execution, execute *free()* to free up space.

3.3.10. ql_mqtt_client_setopt

This function configures the parameter processing type of the MQTT client handle.

● Prototype

```
int ql_mqtt_client_setopt(mqtt_client_t *client, int opt_tag,...)
```

● Parameter

client:

[In] MQTT client handle, which is obtained by *ql_mqtt_client_init()*. See **Chapter 3.3.1** for details.

opt_tag:

[In] Parameter processing type configured in the MQTT client handle context; See **Chapter 3.3.10.1** for details.

● Return Value

See **Chapter 3.3.1.1** for error codes.

3.3.10.1.mqtt_option_e

The enumeration of parameter processing type configured in the MQTT client handle context is defined below:

```
typedef enum{
    MQTT_CLIENT_OPT_REC_BUF_MAX = 1,
    MQTT_CLIENT_OPT_VERSION      = 2,
}mqtt_option_e
```

● Member

Member	Description
<i>MQTT_CLIENT_OPT_REC_BUF_MAX</i>	Maximum downlink data received by MQTT client. Unit: KB. Range: 16–64. Default: 16 KB.
<i>MQTT_CLIENT_OPT_VERSION</i>	MQTT version. Default: 4. 3 MQTT 3.1 4 MQTT 3.1.1

4 Example

Please see MQTT client example file *mqtt_demo.c*, provided in QuecOpen CSDK of EC200U series and EG91xU family modules. The example file is in the *components\ql-application\mqtt* directory.

5 Appendix References

Table 2: Related Documents

Document Name
[1] Quectel_EC200U&EG91xU_Series_QuecOpen(SDK)_Quick_Start_Guide
[2] Quectel_EC200U&EG91xU_Series_QuecOpen(SDK)_Data_Call_API_Reference_Manual

Table 3: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
CA	Certificate Authority
DNS	Domain Name Server
LTE	Long Term Evolution
ID	Identifier
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
QoS	Quality of Service
RTOS	Real-Time Operating System
SDK	Software Development Kit
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol