# QSTM32
# SDK API Design

Version: 2.0

Date: 2025-09-24

State: Preliminary

| Confidentiality Level：(Tick the Box ■) | | |
|---|---|---|
| Top Secret ☐ | Confidential ☐ | Public ■ |

# Document Control Records

| Revision History | | | |
|---|---|---|---|
| Date | Version | Description | Author |
| 2025-09-24 | 2 | Initial version | Wells Li |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**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

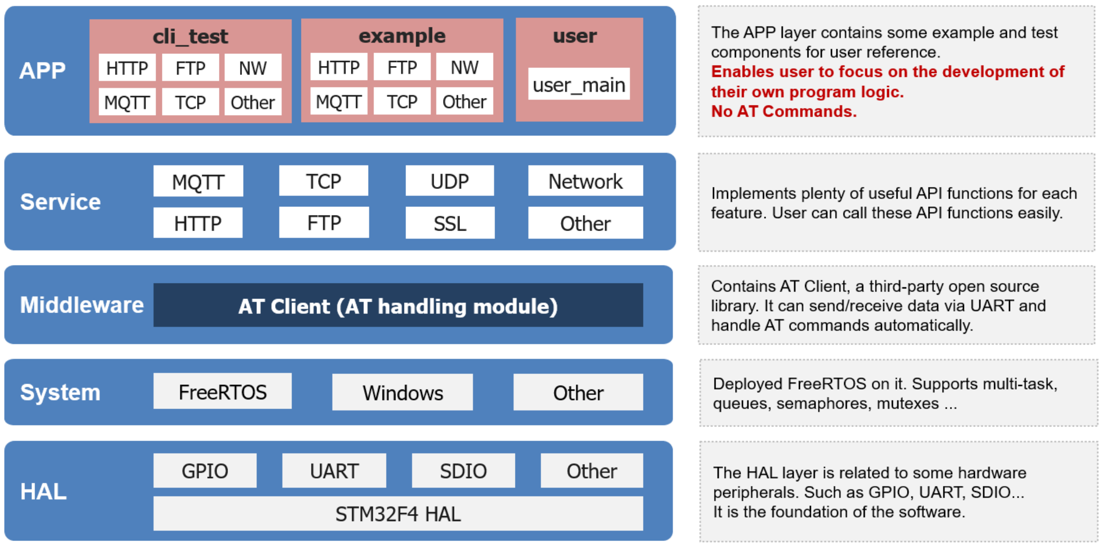
**Or our local offices. For more information, please visit:**
http://www.quectel.com/support/sales.htm.

**For technical support, or to report documentation errors, please visit:**
http://www.quectel.com/support/technical.htm.
Or email us at: support@quectel.com.

# Contents

# 1 Software Framework

A set of software framework, designed by Quectel, provides various interfaces to run AT commands. Thus, the developer can be concentrated on service design without paying much attention on command in lower layer, reducing difficulty in function development heavily.


APP
cli_test
HTTP
FTP
NW
MQTT
TCP
Other
example
HTTP
FTP
NW
MQTT
TCP
Other
user
user_main
The APP layer contains some example and test components for user reference.
Enables user to focus on the development of their own program logic.
No AT Commands.
Service
MQTT
TCP
UDP
Network
HTTP
FTP
SSL
Other
Implements plenty of useful API functions for each feature. User can call these API functions easily.
Middleware
AT Client (AT handling module)
Contains AT Client, a third-party open source library. It can send/receive data via UART and handle AT commands automatically.
System
FreeRTOS
Windows
Other
Deployed FreeRTOS on it. Supports multi-task, queues, semaphores, mutexes ...
HAL
GPIO
UART
SDIO
Other
STM32F4 HAL
The HAL layer is related to some hardware peripherals. Such as GPIO, UART, SDIO...
It is the foundation of the software.

As illustrated above, it is divided into 5 layers

Layer1(Hardware Peripheral): Configure the driver code generated automatically using the STM32CubeIDE tool;

Layer2 (System): Provide OS interface, currently, Freertos is used to develop;

Layer3 (Middleware): Port third-party AT Client components;

Layer4 (Device): Implement API interface of individual module via AT command, including HTTP, FTP, MQTT, Socket and so on;

Layer5 (APP): Some examples of individual module

# 2 Codes Architecture

It is composed by 4 sections.

- App directory

  **example**

  Provides examples of individual module for reference and utilization by client, including FTP, HTTP, Socket and related functions

  **test**

  Test codes utilized by Quectel for function verification and test.

  **user**

  Main directory of user application development. **User_main**: code entry file, which can be used in customized development

- Quectel directory

  Implement AT command, including core codes compiled by Quectel

  **common**

  Inner common public interface to be called by other functions

  **modules**

  Implement AT command, including HTTP, FTP, MQTT and Socket

  **third-party**

  Store third-party components, including **at_client** and **at_socket** currently.

- System directory

  Cross-platform porting, which implements some API interfaces that rely on system, including serial port, semaphore, task and mutex.

- Tools directory

  Store auto compilation script.

# 3 API Introduction

## 3.1. Network Registration

### 3.1.1. Brief Summary

This function provides cellular network registration capability, including initialization, parameter configuration and state management. Moreover, it supports multi-mode compatibility design.

### 3.1.2. Interface Illustration

| Type | Path | Description |
|---|---|---|
| Header | ql_net.h | Interface announcement & Struct definition |
| Source | ql_net.c | Interface implementation codes |
| Example | example_net.c | Quick integration reference |
| Dependency | at_client.c | Initialize AT command framework ahead |

### 3.1.3. Specific Illustration

#### 3.1.3.1. ql_net_init

Function: Network registration initialization
Prototype: ql_net_t ql_net_init(at_client_t client)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |

Returned value

| | |
|---|---|
| Network registration handle | // Success to initialize |
| NULL | // Failure to initialize |

#### 3.1.3.2. ql_usim_get

Function: Query SIM card state
Prototype: QL_NET_ERR_CODE_E ql_usim_get(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

QT_NET_OK // SIM card is normal
Otherwise, it will return other values

#### 3.1.3.3. ql_net_setopt

Function: Configure network registration parameter
Prototype: bool ql_net_setopt(ql_net_t handle, QL_NET_OPTION_E type, …)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |
| type | QL_NET_OPTION_E | see intact enumeration definition in SDK header |
| … | variable parameter | actual value of the parameter |

Returned value

| | |
|---|---|
| True | //Success to configure |
| False | //Failure to configure |

#### 3.1.3.4. ql_net_attach

Function: Network Attach
Prototype: QL_NET_ERR_CODE_E ql_net_attach(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

QT_NET_OK // Success to attach
Otherwise, it will return other values.

#### 3.1.3.5. ql_net_get_signal_info

Function: Query network signal strength and quality
Prototype: int ql_net_get_signal_info(ql_net_t handle, int *strength, int *quality);

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |
| strength | int * | Output parameter, which is used to receive network signal strength |
| quality | int * | Output parameter, which is used to receive network signal quality |

Returned value

0 // Success to query
-1 // Failure to query

#### 3.1.3.6. ql_net_is_ok

Function: Query whether the network is normal
Prototype: int ql_net_is_ok(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

True // Normal
False // Abnormal

#### 3.1.3.7. ql_net_reconnect

Function: Reconnect
Prototype: QL_NET_ERR_CODE_E ql_net_reconnect(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

QT_NET_OK // Success to reconnect
Otherwise, it will return other values

#### 3.1.3.8. ql_module_reboot

Function: Reboot module
Prototype: void ql_module_reboot(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

None

#### 3.1.3.9. ql_net_deinit

Function: Deinitialize
Prototype: void ql_net_deinit(ql_net_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_net_t | ql_net_init returned value |

Returned value

None

## 3.2. HTTP

### 3.2.1. Brief Summary

This function provides HTTP/HTTPS communication capability based on cellular network, supports bilateral data interaction between device and cloud service and adapts CA certificate and bilateral certificate.

### 3.2.2. Interface illustration

| Type | Path | Description |
|---|---|---|
| header | ql_http.h | Interface announcement & Struct definition |
| source | ql_http.c | Interface implementation codes |
| example | example_http.c | Quick integration reference |
| dependency | at_client.c | Initialize AT command framework ahead |

### 3.2.3. Specific Illustration

#### 3.2.3.1. ql_http_init

Function: http interface initializatin
Prototype: ql_http_t ql_http_init(at_client_t client)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |

Returned value

| | |
|---|---|
| HTTP Handle | // Success to initialize |
| NULL | // Failure to initialize |

Note: Even if the interface supports multi-example, the module can only handle one HTTP request at the same time. Upon external call, it is not allowed to call several http requests together

#### 3.2.3.2. ql_http_setopt

Function: http parameter configuration
Prototype: bool ql_http_setopt(ql_http_t handle, QL_HTTP_OPTION_E type, …)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_http_t | ql_http_init returned value |
| type | QL_HTTP_OPTION_E | See intact enumeration definition in SDK header |
| … | variable parameter | actual value |

Returned value

| | |
|---|---|
| True | // Success to configure |
| False | // Failure to configure |

#### 3.2.3.3. ql_http_set_ssl

Function: SSL parameter configuration
Prototype: void ql_http_set_ssl(ql_http_t handle, ql_SSL_Config config)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_http_t | ql_http_init returned value |
| config | ql_SSL_Config | SSL parameter info. See SSL header |

Returned value

None

#### 3.2.3.4. ql_http_request

Function: Send HTTP Request
Prototype: QL_HTTP_ERR_CODE_E ql_http_request(ql_http_t handle, const char* url, QL_HTTP_METHOD_E method, const char* data, size_t data_len)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_http_t | ql_http_init returned value |
| url | const char* | URL address, which supports HTTP(s) |
| method | QL_HTTP_METHOD_E | HTTP request method |
| data | const char* | Request body data pointer |
| data_len | size_t | Request body data length |

Returned value

QT_HTTP_OK //Success to request
Otherwise, it will return other values

Note:
1. This interface is to sync since in stage of http request, any other operation will not be allowed.
2. Generally, the module does not support GET Request with body. As a result, the data in ql_http_request shall be NULL.

Two methods to read data replied by http(s) request.

1. Call ql_htto_recv. When the returned value is smaller than or equal to 0, it means a success to read. However, this method can't set write callback function via ql_http_setopt
2. Set write callback function via ql_http_setopt. Note: In this situation, it is not allowed to call ql_http_recv

Two methods supported by post request

1. Place body data into specific request body pointer. However, it will be applied only under the circumstance that the memory is sufficient.
2. In condition that the insufficient does not satisfy the body size, please set via ql_http_setopt and send body data by packet.

#### 3.2.3.5. ql_http_recv

Function: Read info replied by HTTP Request
Prototype: int ql_http_recv(ql_http_t handle, char* buf, size_t size)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_http_t | ql_http_init returned value |
| buf | char* | Store pointer to reply data, which shall be allocated ahead |
| size | size_t | byte size to be read |

Returned value

Actual length to be read

0 // it finishes reading.

#### 3.2.3.6. ql_http_deinit

Function: HTTP interface deinitialization
Prototype: void ql_http_deinit(ql_http_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_http_t | ql_http_init returned value |

Returned value

None

## 3.3. FTP

### 3.3.1. Brief Summary

This function supports FTP communication capability based on cellular network, file uploading, downloading and folder management between device and server. Additionally, it supports CA certificate.

### 3.3.2. Interface Illustration

| Type | Path | Description |
|---|---|---|
| header | ql_ftp.h | Interface announcement & struct definition |
| source | ql_ftp.c | Interface implementation code |
| example | example_ftp.c | Quick integration reference |
| dependency | at_client.c | Initialize AT command framework ahead |

### 3.3.3. Specific Illustration

#### 3.3.3.1. ql_ftp_init

Function: FTP interface initialization
Prototype: ql_ftp_t ql_ftp_init(at_client_t client)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |

Returned value

| | |
|---|---|
| FTP handle | // Success to initialize |
| NULL | // Failure to initialize |

The function can only support one HTTP request at the same time even if it support multi-example. As a result, when calling externally, please do not login several FTPs simultaneously.

#### 3.3.3.2. ql_ftp_setopt

Function: FTP parameter configuration
Prototype: bool ql_ftp_setopt(ql_ftp_t handle, QL_FTP_OPTION_E type, …)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| type | QL_FTP_OPTION_E | See intact enumeration definition in SDK header |
| … | variable parameter | actual value |

Returned value

| | |
|---|---|
| True | //Success to configure |
| False | // Failure to configure |

#### 3.3.3.3. ql_ftp_set_ssl

Function: ssl parameter configuration
Prototype: bool ql_ftp_set_ssl(ql_ftp_t handle, ql_SSL_Config config)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| config | ql_SSL_Config | SSL parameter info. See SSL header |

Returned value

True //Success to configure
False // Failure to configure

#### 3.3.3.4. ql_ftp_login

Function: Login client terminal
Prototype: QL_FTP_ERR_CODE_E ql_ftp_login(ql_ftp_t handle, const char* username, const char* password)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| username | const char* | authenticate username |
| password | const char* | authenticate password |

Returned value

QT_FTP_OK // Success to login
Or it will return other values

#### 3.3.3.5. ql_ftp_cwd

Function: Set working directory
Prototype: QL_FTP_ERR_CODE_E ql_ftp_cwd(ql_ftp_t handle, const char* path)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | working directory path |

Returned value

QT_FTP_OK // Success to set
Or it will return other values

#### 3.3.3.6. ql_ftp_pwd

Function: query current working directory
Prototype: QL_FTP_ERR_CODE_E ql_ftp_pwd(ql_ftp_t handle, char* path)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | Pointer to store current working directory working path, which shall be allocated ahead. |

Returned value

QT_FTP_OK // Success to query
Or it will return other values

#### 3.3.3.7. ql_ftp_mkdir

Function: Create remote folder

Prototype: QL_FTP_ERR_CODE_E ql_ftp_mkdir(ql_ftp_t handle, const char* path)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | needed to build remote directory path |

Returned value

QT_FTP_OK // Success to create
Or it will return other values

#### 3.3.3.8. ql_ftp_rmdir

Function: Remove remote directory
Prototype: QL_FTP_ERR_CODE_E ql_ftp_rmdir(ql_ftp_t handle, const char* path)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | remove remote directory path |

Returned value

QT_FTP_OK // Success to remove
Or it will return other values

#### 3.3.3.9. ql_ftp_rename

Function: Rename file/folder
Prototype: QL_FTP_ERR_CODE_E ql_ftp_rename(ql_ftp_t handle, const char* old_name, const char* new_name)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| old_name | const char* | Path of raw file/folder |
| new_name | const char* | Path of new file/folder |

Returned value

QT_FTP_OK // Success to rename
Or it will return other values

#### 3.3.3.10. ql_ftp_list

Function: Query all files/folders info in designated directory
Prototype: QL_FTP_ERR_CODE_E ql_ftp_list(ql_ftp_t handle, const char *path, ql_ftp_file_info_s **head)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | Query directory path |
| head | ql_ftp_file_info_s ** | address to store file info chain header point, which demands calling ql_ftp_list_free externally. For struct entity, see header. |

Returned value

QT_FTP_OK // Success to query
Or it will return other values

#### 3.3.3.11. ql_ftp_nlist

Function: Query all files/folders name in designated directory
Prototype: QL_FTP_ERR_CODE_E ql_ftp_nlist(ql_ftp_t handle, const char *path, ql_ftp_file_info_s **head)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| path | const char* | Query directory path |
| head | ql_ftp_file_info_s ** | address to store file info chain header point, which demands calling ql_ftp_list_free externally. For struct entity, see header. |

Returned value

QT_FTP_OK // Success to query
Or it will return other values

#### 3.3.3.12. ql_ftp_list_free

Function: Free memory space allocated by ql_ftp_list or ql_ftp_nlist
Prototype: void ql_ftp_list_free(ql_ftp_file_info_s *head)

| Parameter | Type | Description |
|---|---|---|
| head | ql_ftp_file_info_s * | Memory space allocated by ql_ftp_list or ql_ftp_nlist |

Returned value

None

#### 3.3.3.13. ql_ftp_file_size

Function: Get designated file size
Prototype:QL_FTP_ERR_CODE_E ql_ftp_file_size(ql_ftp_t handle, const char *remote_file_name, size_t *file_size)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| remote_file_name | const char* | file name |
| file_size | size_t * | Output parameter. Once a success, it will write file size. |

Returned value

QT_FTP_OK // Success to query
Or it will return other values

### 3.3.3.14. ql_ftp_upload

Function: Upload local file to remote server
Prototype: QL_FTP_ERR_CODE_E ql_ftp_upload(ql_ftp_t handle, const char *local_file_name, const char *remote_file_name)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| local_file_name | const char* | local file path |
| remote_file_name | const char* | remote server file path |

Returned value

QT_FTP_OK // Success to upload
Or it will return other values

### 3.3.3.15. ql_ftp_download

Function: Download file to local path from remote server

Prototype: QL_FTP_ERR_CODE_E ql_ftp_download(ql_ftp_t handle, const char *remote_file_name, const char *local_file_name)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| remote_file_name | const char* | remote server file path |
| local_file_name | const char* | local file path |

Returned value

QT_FTP_OK // Success to download
Or it will return other values

### 3.3.3.16. ql_ftp_file_delete

Function: Delete file
Prototype: QL_FTP_ERR_CODE_E ql_ftp_file_delete(ql_ftp_t handle, const char *remote_file_name)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| remote_file_name | const char* | remote server file path |

Returned value

QT_FTP_OK // Success to download
Or it will return other values

#### 3.3.3.17. ql_ftp_file_get_modify_time

Function: Query the last time when the file is modified.
Prototype: QL_FTP_ERR_CODE_E ql_ftp_file_get_modify_time(ql_ftp_t handle, const char *remote_file_name, char *time)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |
| remote_file_name | const char* | file name |
| time | char* | address to store the last time when the file is modified in a format of YYYYMMDDHHMMSS or YYYYMMDDHHMMSS.NNN |

Returned value

QT_FTP_OK // Success to query
Or it will return other values

#### 3.3.3.18. ql_ftp_logout

Function: Logout client terminal
Prototype: QL_FTP_ERR_CODE_E ql_ftp_logout(ql_ftp_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |

Returned value

QT_FTP_OK // Success to logout
Or it will return other values

#### 3.3.3.19. ql_ftp_uninit

Function: ftp uninitialization
Prototype: void ql_ftp_uninit(ql_ftp_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_ftp_t | ql_ftp_init returned value |

Returned value

None

## 3.4. MQTT

### 3.4.1. Brief Summary

This kind of function provides IoT communication capability based on MQTT, supports secure connection, message publish/subscribe and Qos control between device and MQTT Broker. In addition, it supports TLS/SSL and CA, guaranteeing data transmission safety.

### 3.4.2. Interface Illustration

| Type | Path | Description |
|---|---|---|
| header | ql_mqtt.h | Interface announcement & struct entity definition |
| source | ql_mqtt.c | Interface implementation codes |
| example | example_mqtt.c | Quick integration reference |
| dependency | at_client.c | Initialize AT command framework ahead |

### 3.4.3. Specific Illustration

#### 3.4.3.1. ql_mqtt_create

Function: Create MQTT client handle
Prototype：ql_mqtt_t ql_mqtt_create (at_client_t client)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |

Returned value

| | |
|---|---|
| MQTT handle | //Success to initialize |
| NULL | // Failure to initialize |

Note: At maximum 6 MQTT examples are supported

#### 3.4.3.2. ql_mqtt_setopt

Function: mqtt parameter configuration
Prototype: bool ql_mqtt_setopt(ql_mqtt_t handle, QL_MQTT_OPTION_E type, …)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| type | QL_MQTT_OPTION_E | See intact enumeration definition in SDK header |
| … | variable parameter | actual value |

Returned value

| | |
|---|---|
| True | // Success to configure |
| False | // Failure to configure |

#### 3.4.3.3. ql_mqtt_set_ssl

Function: ssl parameter configuration
Prototype: bool ql_mqtt_set_ssl(ql_mqtt_t handle, ql_SSL_Config config)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| config | ql_SSL_Config | SSL parameter info. See SSL header |

Returned value

| | |
|---|---|
| True | // Success to configure |
| False | // Failure to configure |

#### 3.4.3.4. ql_mqtt_connect

Function: Connect MQTT server
Prototype: QL_MQTT_ERR_CODE_E ql_mqtt_connect(ql_mqtt_t handle, const char* server, int port, const char* username, const char* password)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| server | const char* | MQTT server address |
| port | int | MQTT server port |
| username | const char* | Authenticate username. If none, NULL will be uploaded |
| password | const char* | Authenticate password. If none, NULL will be uploaded |

Returned value

QL_MQTT_OK //Success to connect
Or it will return other values

#### 3.4.3.5. ql_mqtt_disconnect

Function: Disconnect server
Prototype: void ql_mqtt_disconnect(ql_mqtt_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |

Returned value

None

#### 3.4.3.6. ql_mqtt_pub

Function: Publish topic message
Prototype: QL_MQTT_ERR_CODE_E ql_mqtt_pub(ql_mqtt_t handle, const char *topic, const char *message, QL_MQTT_QOS_LEVEL_E qos, bool retain)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| topic | const char * | Publish topic |
| message | const char * | message content |
| qos | QL_MQTT_QOS_LEVEL_E | QoS |
| retain | bool | Retain message or not |

Returned value

QT_MQTT_OK // Success to publish
Or it will return other values

#### 3.4.3.7. ql_mqtt_sub

Function: Subscribe topic
Prototype: QL_MQTT_ERR_CODE_E ql_mqtt_sub(ql_mqtt_t handle, const char *topic, QL_MQTT_QOS_LEVEL_E qos)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| topic | const char * | subscribe topic |
| qos | QL_MQTT_QOS_LEVEL_E | Qos |

Returned value

QT_MQTT_OK //Success to subscribe
Or it will return other values

#### 3.4.3.8. ql_mqtt_unsub

Function: Unsubscribe topic
Prototype: QL_MQTT_ERR_CODE_E ql_mqtt_unsub(ql_mqtt_t handle, const char *topic)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |
| topic | const char * | Subscribe topic |

Returned value

QT_MQTT_OK // Success to unsubscribe
Or it will return other values

#### 3.4.3.9. ql_mqtt_destroy

Function: Destroy MQTT handle
Prototype: void ql_mqtt_destroy(ql_mqtt_t handle)

| Parameter | Type | Description |
|---|---|---|
| handle | ql_mqtt_t | ql_mqtt_init returned value |

Returned value

None

## 3.5. SOCKET

### 3.5.1. Brief Summary

This function provides network communication capability based on TCP/UDP Socket and supports stable data transmission between device and server, which can be applied in scenario demanding low latency and high reliability.

### 3.5.2. Interface Illustration

| Type | Path | Description |
|---|---|---|
| header | ql_socket.h | Socket announcement & Struct definition |
| source | ql_socket.c | Interface implementation codes |
| example | socketdirectory | Quick integration reference |
| dependency | at_client.c | Initialize AT command framework ahead |

### 3.5.3. Specific Illustration

#### 3.5.3.1. Socket

Function: Build one socket descriptor
Prototype: int socket (int domain, int type, int protocol)

| Parameter | Type | Description |
|---|---|---|
| domain | int | Designate communication domain/protocol. Currently, it only supports AF_INET(IPv4) |
| type | int | Designate Socket type. Currently, it only supports SOCK_STREAM(TCP) and SOCK_DGRAM(UDP) |

Returned value

Nonnegative integer // Success to build
-1 // Failure to build

Note: at maximum 12 socket descriptors can be supported at the same time.

#### 3.5.3.2. Close

Function: Close file descriptor
Prototype: int close (int sockfd)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |

Returned value

0 // Success to close
-1 // Failure to close

#### 3.5.3.3. Shutdown

Function: Shutdown file descriptor
Prototype: int shutdown(int sockfd, int how)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| how | int | Compatible, which is vacant actually |

Returned value

0 //Success to shutdown
-1 //Failure to shutdown

#### 3.5.3.4. Bind

Function: Bind socket to designated IP address and port number
Prototype: int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | Socket descriptor |
| addr | const struct sockaddr* | IP to be bound (any IP is OK, which will be bound to 127.0.0.1 internally) and port number |
| addrlen | socklen_t | Addr struct length |

Returned value

0 //Success to bind
-1 //Failure to bind

#### 3.5.3.5. Connect

Function: The client connects to server actively via TCP/UDP socket
Prototype: int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| addr | const struct sockaddr* | Point at struct of server address, including server IP and port info |
| addrlen | socklen_t | addr struct length |

Returned value

0 //Success to connect
-1 //Failure to connect

#### 3.5.3.6. Sendto

Function: Send data to designated target address via UDP
Prototype: int sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| buf | const void* | buffer of data to be sent |
| len | size_t | data length |
| flags | int | Control sending, which is 0 by default |
| to | const struct sockaddr * | Target address |

| Parameter | Type | Description |
|---|---|---|
| tolen | socklen_t | actual length of to |

Returned value

Data length has been sent // Success to send
-1 // Failure to send

#### 3.5.3.7. Send

Function: Send data to designated target address via TCP
Prototype: int send(int sockfd, const void *buf, size_t len, int flags)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| buf | const void* | buffer of data to be sent |
| len | size_t | data length |
| flags | int | Control sending, which is 0 by default |
| to | const struct sockaddr * | Target address |
| tolen | socklen_t | actual length of to |

Returned value

Data length has been sent // Success to send
-1 // Failure to send

#### 3.5.3.8. Recvfrom

Function: Receive data via UDP and get its IP
Prototype: int recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| buf | void* | buffer to receive data |
| len | size_t | maximum length of buffer |
| flags | int | Control receiving, which is 0 by default |
| from | struct sockaddr * | Store sender address |
| fromlen | socklen_t | Upon inputting, it indicates the buffer length of from. Upon outputting, it indicates the actual address length |

Returned value

```
Data length has been received    // Success to receive
-1                               // Failure to receive
```

#### 3.5.3.9. Recv

Function: Receive data via TCP
Prototype: int ql_recv(int sockfd, void *buf, size_t len, int flags)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| buf | void* | buffer to receive data |

| Parameter | Type | Description |
|---|---|---|
| len | size_t | maximum length of buffer |
| flags | int | Control receiving, which is 0 by default |

Returned value

| | |
|---|---|
| Data length has been received | // Success to return |
| -1 | // Failure to return |

### 3.5.3.10. Setsockopt

Function: Set socket option (Currently, it only supports setting timeout duration)
Prototype: int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| level | int | level of option definition |
| optname | int | option name to be set |
| optval | const void* | Ponit at buffer that contains new option value |
| optlen | socklen_t | optval buffer size |

Returned value

| | |
|---|---|
| 0 | // Success to set |
| -1 | // Failure to set |

### 3.5.3.11. Listen

Function: Set the Socket as listened mode to wait connection request from client side

Prototype: int listen(int sockfd, int backlog)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| backlog | int | maximum length of waiting queue to be connected |

Returned value

0 // Success to listen
-1 // Failure to listen

### 3.5.3.12. Accept

Function: Accept one client connection request and return one new socket descriptor
Prototype: int accept(int sockfd, struct sockaddr *name, socklen_t *namelen)

| Parameter | Type | Description |
|---|---|---|
| sockfd | int | socket descriptor |
| name | struct sockaddr * | store client address info |
| namelen | socklen_t* | Upon inputting, it will designate buffer size of Name.<br>Upon reciving, it will store actual address length |

Returned value

Nonnegative integer value // Success to accept
-1 // Failure to accept

### 3.5.3.13. Select

Function: Listen to several file descriptors to check whether they are readable, implementing several I/O operations via single thread.
Prototype: int select(int nfds, ql_fd_set *readfds, ql_fd_set *writefds, ql_fd_set *exceptfds, struct timeval *timeout)

| Parameter | Type | Description |
|---|---|---|
| nfds | int | Maximum file descriptor value +1 |
| readfds | ql_fd_set* | Check readable file descriptor aggregation |
| writefds | ql_fd_set* | Set as NULL |
| exceptfds | ql_fd_set* | Set as NULL |
| timeout | struct timeval * | Timeout（NULL: congestion; 0: return immediately） |

Returned value

```
> 0     //Ready file descriptor quantity
=0      //Timeout
<0      // Error
```

## 3.6. PSM

### 3.6.1. Brief Summary

This function provides management interface related to power saving mode, including setting PSM and sleep control.

### 3.6.2. Interface Illustration

| Type | Path | Description |
|---|---|---|
| header | ql_psm.h | Interface announcement & struct definition |
| source | ql_psm.c | Interface implementation codes |
| example | example_psm.c | Quick integration reference |

| Type | Path | Description |
|---|---|---|
| dependency | at_client.c | Initialize AT command framework ahead |

## 3.6.3. Specific Illustration

### 3.6.3.1. ql_psm_settings_write

Function: PSM parameter setting
Prototype: int ql_psm_settings_write(at_client_t client, ql_psm_setting_s settings)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |
| settings | ql_psm_setting_s | PSM-related parameter to be set. See header for struct |

Returned value

0 //Success to set
-1 // Failure to set

### 3.6.3.2. ql_psm_settings_read

Function: Read psm parameter
Prototype: int ql_psm_settings_read(at_client_t client, ql_psm_setting_s *settings)

| Parameter | Type | Description |
|---|---|---|
| client | at_client_t | AT command handle in client side.<br>Single-mode: at_client_get_first().<br>Multi-mode: specific example shall be assigned |
| settings | ql_psm_setting_s* | Point at pointer of ql_psm_setting_s, which will be used to store the PSM configuration parameters have been read |

Return value

```
0      // Success to read
-1    // Failure to read
```