

# 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](mailto: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](mailto: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 functions 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 functions 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. 2025. All rights reserved.**

# Contents

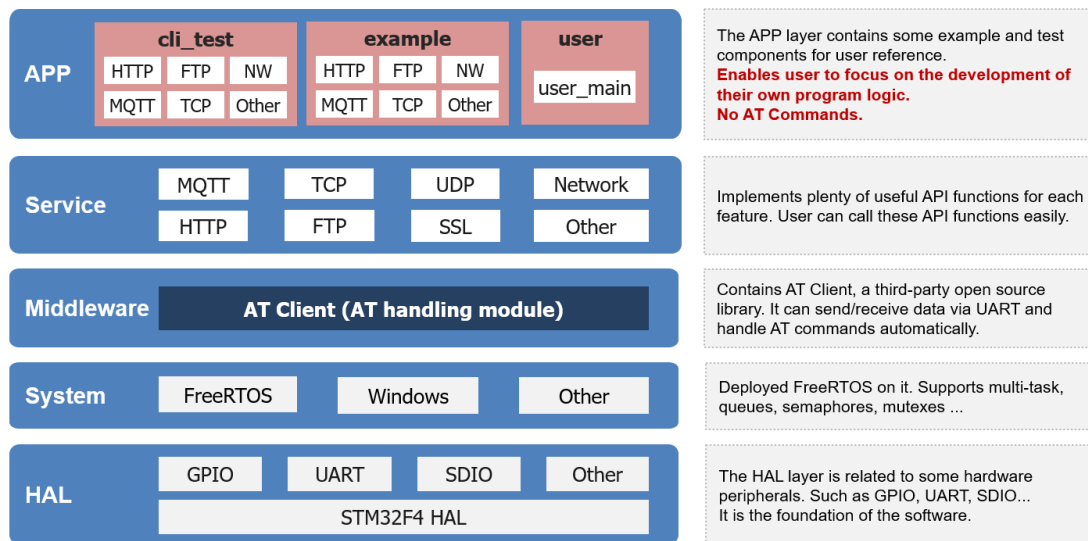
<b>Document Control Records .....</b>	<b>1</b>
<b>Contents .....</b>	<b>4</b>
<b>1 Software Framework.....</b>	<b>7</b>
<b>2 Codes Architecture.....</b>	<b>8</b>
<b>3 API Introduction .....</b>	<b>9</b>
3.1. Network Registration.....	9
3.1.1. Brief Summary .....	9
3.1.2. Interface Illustration.....	9
3.1.3. Specific Illustration .....	9
3.1.3.1. ql_net_init.....	9
3.1.3.2. ql_usim_get.....	10
3.1.3.3. ql_net_setopt.....	10
3.1.3.4. ql_net_attach.....	11
3.1.3.5. ql_net_get_signal_info .....	11
3.1.3.6. ql_net_is_ok.....	11
3.1.3.7. ql_net_reconnect.....	12
3.1.3.8. ql_module_reboot .....	12
3.1.3.9. ql_net_deinit.....	13
3.2. HTTP.....	13
3.2.1. Brief Summary .....	13
3.2.2. Interface illustration.....	13
3.2.3. Specific Illustration .....	14
3.2.3.1. ql_http_init.....	14
3.2.3.2. ql_http_setopt.....	14
3.2.3.3. ql_http_set_ssl .....	15
3.2.3.4. ql_http_request.....	15
3.2.3.5. ql_http_recv.....	16
3.2.3.6. ql_http_deinit.....	17
3.3. FTP.....	17
3.3.1. Brief Summary .....	17
3.3.2. Interface Illustration.....	17
3.3.3. Specific Illustration .....	18
3.3.3.1. ql_ftp_init.....	18
3.3.3.2. ql_ftp_setopt.....	18
3.3.3.3. ql_ftp_set_ssl .....	19
3.3.3.4. ql_ftp_login .....	19
3.3.3.5. ql_ftp_cwd .....	20
3.3.3.6. ql_ftp_pwd .....	20
3.3.3.7. ql_ftp_mkdir.....	20

3.3.3.8.	ql_ftp_rmdir .....	21
3.3.3.9.	ql_ftp_rename .....	21
3.3.3.10.	ql_ftp_list .....	22
3.3.3.11.	ql_ftp_nlist .....	23
3.3.3.12.	ql_ftp_list_free .....	23
3.3.3.13.	ql_ftp_file_size .....	23
3.3.3.14.	ql_ftp_upload .....	24
3.3.3.15.	ql_ftp_download .....	24
3.3.3.16.	ql_ftp_file_delete .....	25
3.3.3.17.	ql_ftp_file_get_modify_time .....	26
3.3.3.18.	ql_ftp_logout .....	26
3.3.3.19.	ql_ftp_uninit .....	27
3.4.	MQTT .....	27
3.4.1.	Brief Summary .....	27
3.4.2.	Interface Illustration .....	27
3.4.3.	Specific Illustration .....	28
3.4.3.1.	ql_mqtt_create .....	28
3.4.3.2.	ql_mqtt_setopt .....	28
3.4.3.3.	ql_mqtt_set_ssl .....	29
3.4.3.4.	ql_mqtt_connect .....	29
3.4.3.5.	ql_mqtt_disconnect .....	30
3.4.3.6.	ql_mqtt_pub .....	30
3.4.3.7.	ql_mqtt_sub .....	31
3.4.3.8.	ql_mqtt_unsub .....	31
3.4.3.9.	ql_mqtt_destroy .....	32
3.5.	SOCKET .....	32
3.5.1.	Brief Summary .....	32
3.5.2.	Interface Illustration .....	32
3.5.3.	Specific Illustration .....	33
3.5.3.1.	Socket .....	33
3.5.3.2.	Close .....	33
3.5.3.3.	Shutdown .....	34
3.5.3.4.	Bind .....	34
3.5.3.5.	Connect .....	35
3.5.3.6.	Sendto .....	35
3.5.3.7.	Send .....	36
3.5.3.8.	Recvfrom .....	37
3.5.3.9.	Recv .....	37
3.5.3.10.	Setsockopt .....	38
3.5.3.11.	Listen .....	38
3.5.3.12.	Accept .....	39
3.5.3.13.	Select .....	39
3.6.	PSM .....	40
3.6.1.	Brief Summary .....	40

3.6.2.	Interface Illustration.....	40
3.6.3.	Specific Illustration .....	41
3.6.3.1.	ql_psm_settings_write.....	41
3.6.3.2.	ql_psm_settings_read.....	41

# 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.



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. Single-mode: at_client_get_first(). 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. Single-mode: at_client_get_first(). 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\_http\_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. Single-mode: at_client_get_first(). 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. Single-mode: at_client_get_first(). 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
-----------	------	-------------

to                      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

to                                      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. 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. Single-mode: at_client_get_first(). 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. Single-mode: at_client_get_first(). 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