EC200U Series QuecOpen SSL API Reference Manual

LTE Standard Module Series

Version: 1.0

Date: 2022-03-28

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. 2022. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2021-02-24	Kruskal ZHU	Creation of the document
1.0	2022-03-28	Kruskal ZHU/ Charis JIANG	First official release

Contents

Table Index

1 Introduction

Quectel LTE Standard EC200U series module supports QuecOpen® solution. QuecOpen® is an embedded development platform based on RTOS. It is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see ***document [1]***.

This document introduces SSL API, calling process and example in application of Quectel EC200U series module in QuecOpen® solution.

2 SSL API Calling Process

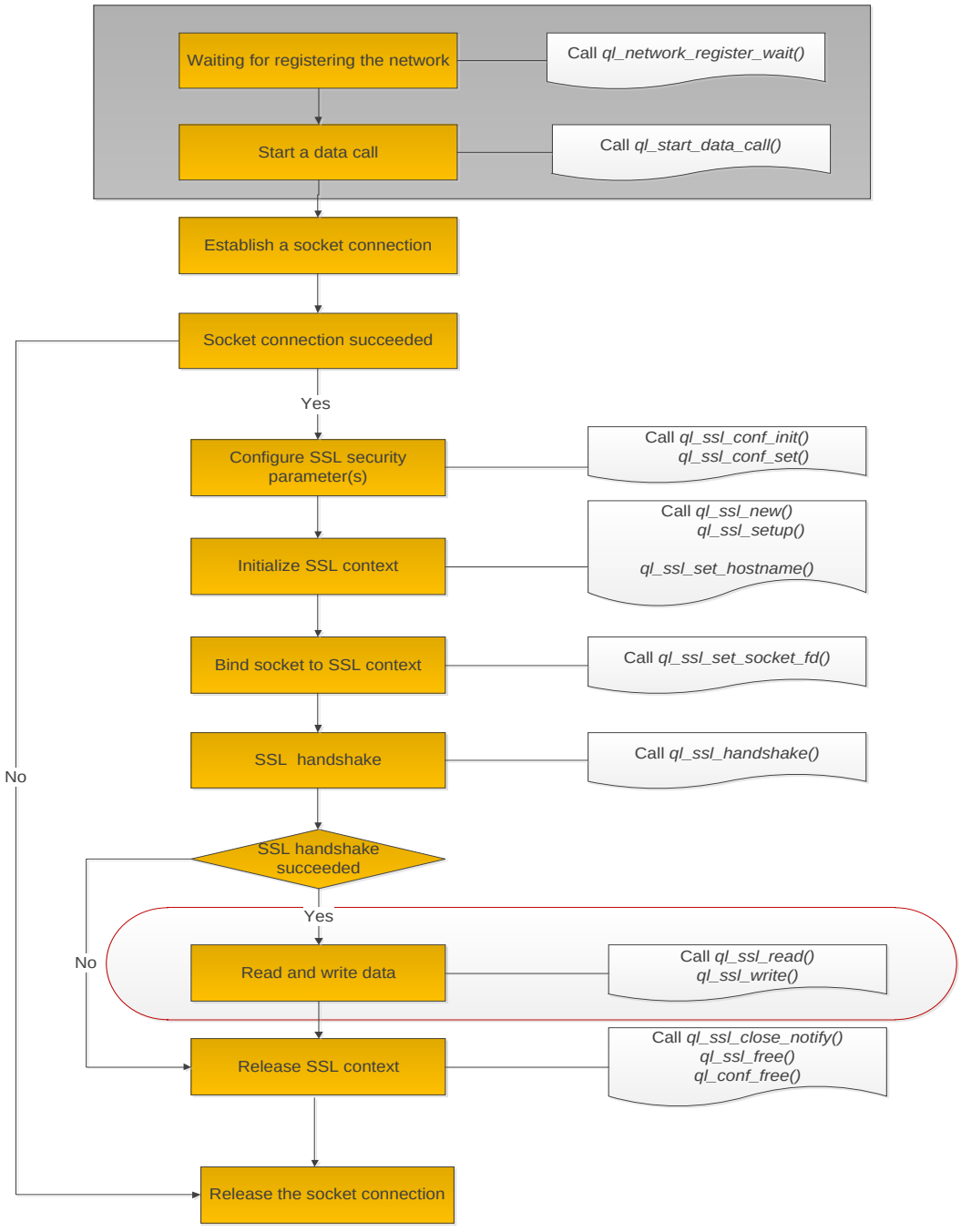

Figure 1: SSL API Calling Process

Before establishing an SSL session connection, you need to register the network and start a data call to establish a data channel (as shown in the gray part of the figure above), see ***document [3]*** for details. If the data channel has been established in other tasks, you can skip this step.

SSL session is established based on the socket connection, so you need to create a socket, bind the activated data channel IP and establish a socket connection before the SSL handshake. After the socket connection is established successfully, you need to configure the SSL security parameters, initialize SSL context and bind the corresponding socket descriptor to the specified SSL context. After the SSL handshake is successful, the data can be read and written. The SSL context and socket connection can be released after the data is read and written.

3 SSL API

3.1. Header File

ql_ssl.h, the header file of SSL API, is located in the *components\ql-kernel\inc* directory. Unless otherwise specified, all header files mentioned in this document are all located in this directory.

3.2. API Overview

Table 1: API Overview

Function	Description
ql_ssl_conf_init()	Initializes SSL default configuration parameters.
ql_ssl_conf_set()	Sets the specified SSL configuration parameters.
ql_ssl_conf_get()	Gets the specified SSL configuration parameters.
ql_ssl_conf_set_by_id()	Sets SSL context configuration parameters corresponding to SSL context ID.
ql_ssl_conf_get_by_id()	Gets SSL context configuration parameters corresponding to SSL context ID.
ql_ssl_conf_free()	Releases the resources occupied by SSL configuration parameters.
ql_ssl_new()	Generates a new SSL context.
ql_ssl_setup()	Sets an SSL context.
ql_ssl_set_socket_fd()	Binds socket descriptor and SSL context.
ql_ssl_set_hostname()	Sets the domain name of the server and is only valid when SNI is set.
ql_ssl_handshake()	Performs the SSL handshake.

ql_ssl_close_notify()	Notifies the server that the SSL connection is closing.
ql_ssl_get_bytes_avail()	Gets the length of readable data from the data buffer in SSL context.
ql_ssl_read()	Reads the readable data from the data buffer in SSL context.
ql_ssl_write()	Sends data to the server through SSL context.
ql_ssl_free()	Releases the resources occupied by the SSL context.
ql_ssl_handshake_finished()	Queries whether SSL handshake has finished.
ql_ssl_ciphersuit_is_valid()	Queries whether the cipher suit ID is valid.

3.3. API Description

3.3.1. ql_ssl_conf_init

This function initializes SSL default configuration parameters.

- **Prototype**

```
int    ql_ssl_conf_init(ql_ssl_config *conf)
```

- **Parameter**

conf:
[In] SSL configuration handle. See ***Chapter 3.3.3.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.1.1. ql_ssl_error_code_e

The SSL result codes indicate whether the function is executed successfully or not, and the enumeration of SSL result codes:

```
typedef enum{
    QL_SSL_SUCCESS                  = 0,
    QL_SSL_ERROR_UNKOWN             = -1,
```

```
    QL_SSL_ERROR_WOUNDBLOCK          = -2,
    QL_SSL_ERROR_INVALID_PARAM      = -3,
    QL_SSL_ERROR_OUT_OF_MEM          = -4,
    QL_SSL_ERROR_NOT_SUPPORT         = -5,
    QL_SSL_ERROR_HS_FAILURE          = -6,
    QL_SSL_ERROR_DECRYPT_FAILURE    = -7,
    QL_SSL_ERROR_ENCRYPT_FAILURE    = -8,
    QL_SSL_ERROR_HS_INPROGRESS       = -9,
    QL_SSL_ERROR_BAD_REQUEST         = -10,
    QL_SSL_ERROR_WANT_READ           = -11,
    QL_SSL_ERROR_WANT_WRITE          = -12,
    QL_SSL_ERROR_SOCKET_RESET        = -13,
}ql_ssl_error_code_e
```

- **Member**

Member	Description
QL_SSL_SUCCESS	Successful execution.
QL_SSL_ERROR_UNKOWN	Unknown error.
QL_SSL_ERROR_WOUNDBLOCK	The operation is not completed and the result is waiting for asynchronous notification.
QL_SSL_ERROR_INVALID_PARAM	Invalid parameter(s).
QL_SSL_ERROR_OUT_OF_MEM	Out of memory.
QL_SSL_ERROR_NOT_SUPPORT	The operation is not supported.
QL_SSL_ERROR_HS_FAILURE	SSL handshake failed.
QL_SSL_ERROR_DECRYPT_FAILURE	Failed to decrypt.
QL_SSL_ERROR_ENCRYPT_FAILURE	Failed to encrypt.
QL_SSL_ERROR_HS_INPROGRESS	SSL handshake is in process.
QL_SSL_ERROR_BAD_REQUEST	Request error.
QL_SSL_ERROR_WANT_READ	No readable data.
QL_SSL_ERROR_WANT_WRITE	No writable data.
QL_SSL_ERROR_SOCKET_RESET	Socket disconnected abnormally.

3.3.2. ql_ssl_conf_set

This function sets the specified SSL configuration parameters.

- **Prototype**

```
int    ql_ssl_conf_set(ql_ssl_config *conf, int type, ...)
```

- **Parameter**

conf:
[In] SSL configuration handle. See ***Chapter 3.3.3.1*** for details.

type:
[In] SSL parameter configuration type. See ***Chapter 3.3.2.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.2.1. ql_ssl_config_type_e

The enumeration of SSL parameter configuration types:

```
typedef enum{
    QL_SSL_CONF_VERSION            = 1,
    QL_SSL_CONF_TRANSPORT          = 2,
    QL_SSL_CONF_CIPHERSUITE        = 3,
    QL_SSL_CONF_AUTHMODE           = 4,
    QL_SSL_CONF_CACERT             = 5,
    QL_SSL_CONF_OWNCERT            = 6,
    QL_SSL_CONF_SNI                = 7,
    QL_SSL_CONF_HS_TIMEOUT         = 8,
    QL_SSL_CONF_IGNORE_LOCALTM = 9,
    QL_SSL_CONF_HS_TIMEOUT_FUNC=  10,
    QL_SSL_CONF_IGNORE_INVALID_CERT_SIGN  =  11,
    QL_SSL_CONF_IGNORE_CERT_ITEM          =  12,
    QL_SSL_CONF_IGNORE_MULTI_CERTCHAIN_VERIFY   =  13
    #ifdef  QL_SSL_TLS_SESSION_CACHE_FEATURE
    QL_SSL_CONF_SESSION_CACHE   =  14,
    #endif
    QL_SSL_CONF_CACERT_BUFFER   =  15,
    QL_SSL_CONF_OWNCERT_BUFFER     = 16,
}ql_ssl_config_type_e
```

- **Member**

Member	Description
QL_SSL_CONF_VERSION	Sets SSL version. See ***Chapter 3.3.2.2*** for details.
QL_SSL_CONF_TRANSPORT	Sets SSL communication method. See ***Chapter 3.3.2.3*** for details.
QL_SSL_CONF_CIPHERSUITE	Sets SSL cipher suite.
QL_SSL_CONF_AUTHMODE	Set SSL verification method. See ***Chapter 3.3.2.4*** for details.
QL_SSL_CONF_CACERT	Sets SSL CA certificate list.
QL_SSL_CONF_OWNCERT	Sets SSL local certificate.
QL_SSL_CONF_SNI	Sets whether to enable SSL SNI.
QL_SSL_CONF_HS_TIMEOUT	Sets the SSL handshake timeout.
QL_SSL_CONF_IGNORE_LOCALTM	Sets whether to ignore local time.
QL_SSL_CONF_HS_TIMEOUT_FUNC	Sets the timeout manipulation function of SSL handshake.
QL_SSL_CONF_IGNORE_INVALID_CERT_SIGN	Sets whether to ignore the check of SSL certificate sent by the server.
QL_SSL_CONF_IGNORE_CERT_ITEM	Sets whether to ignore the check of SSL certificate items sent by the server.
QL_SSL_CONF_IGNORE_MULTI_CERTCHAIN_VERIFY	Sets whether to ignore SSL multi-level certificate chain verification.
QL_SSL_CONF_SESSION_CACHE	Sets whether to enable SSL session reuse function.
QL_SSL_CONF_CACERT_BUFFER	Sets buffer size of SSL CA certificate.
QL_SSL_CONF_OWNCERT_BUFFER	Sets buffer size of SSL local certificate.

3.3.2.2. ql_ssl_version_type_e

The enumeration of SSL versions:

```
typedef enum
{
    QL_SSL_VERSION_0 = 0,
    QL_SSL_VERSION_1,
    QL_SSL_VERSION_2,
```

```
    QL_SSL_VERSION_3,
    QL_SSL_VERSION_ALL
} ql_ssl_version_type_e
```

- **Member**

Member	Description
QL_SSL_VERSION_0	SSL 3.0 version
QL_SSL_VERSION_1	TLS 1.0 version (SSL 3.1)
QL_SSL_VERSION_2	TLS 1.1 version (SSL 3.2)
QL_SSL_VERSION_3	TLS 1.2 version (SSL 3.3)
QL_SSL_VERSION_ALL	All SSL versions

3.3.2.3. ql_ssl_transport_type_e

The enumeration of SSL communication methods:

```
typedef enum{
    QL_SSL_TLS_PROTOCOL    = 0,
    QL_SSL_DTLS_PROTOCOL = 1,
}ql_ssl_transport_type_e
```

- **Member**

Member	Description
QL_SSL_TLS_PROTOCOL	TLS, based on TCP socket communication.
QL_SSL_DTLS_PROTOCOL	DTLS, based on UDP socket communication.

3.3.2.4. ql_ssl_authmode_e

The enumeration of SSL verification methods:

```
typedef enum
{
    QL_SSL_VERIFY_NULL          = 0x0000,
    QL_SSL_VERIFY_SERVER        = 0x0001,
```

```
    QL_SSL_VERIFY_CLIENT_SERVER = 0x0002,
} ql_ssl_authmode_e
```

- **Member**

Member	Description
QL_SSL_VERIFY_NULL	When the server does not require client verification, this option indicates that the client does not verify the server. When the server requires client verification, setting this option causes the SSL handshake to fail.
QL_SSL_VERIFY_SERVER	When the server does not require client verification, this option indicates that the client needs to verify the server. When the server requires client verification, setting this option causes the SSL handshake to fail.
QL_SSL_VERIFY_CLIENT_SERVER	When the server does not require client verification, this option is equivalent to *QL_SSSL_VERIFY_SERVER*. When the server requires client verification, this option must be set.

3.3.3. ql_ssl_conf_get

This function gets the specified SSL configuration parameters.

- **Prototype**

```
int    ql_ssl_conf_get(ql_ssl_config *conf, int type, ...)
```

- **Parameter**

conf:
[In] SSL attribute configuration. See ***Chapter 3.3.3.1*** for details.

type:
[In] SSL parameter configuration type. See ***Chapter 3.3.2.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.3.1. ql_ssl_config

The structure of SSL attribute configurations:

```
typedef struct{
    int                                 ssl_version;
    int                                 transport;
    int                                 *ciphersuites;
    int                                 auth_mode;
    int                                 sni_enable;
    char                                *ca_cert_path[QL_MAX_CA_CERT_CNT];
    char                                *own_cert_path;
    char                                *own_key_path;
    char                                *own_key_pwd;
    char                                *ca_cert_buffer[QL_MAX_CA_CERT_CNT];
    int                                 ssl_negotiate_timeout;
    ql_ssl_handshake_timeout_cb         negotiate_timeout_cb;
    void                                *negotiate_timeout_cb_arg;
    int                                 ignore_invalid_certsign;
    uint32_t                            ignore_certitem;
    int                                 ignore_multi_certchain_verify;
    bool                                client_cert_type;
    #ifdef  QL_SSL_TLS_SESSION_CACHE_FEATURE
    ql_ssl_session                      ssl_session_cache;
    #endif
}ql_ssl_config
```

- **Parameter**

Type	Parameter	Description
int	*ssl_version*	SSL version.
int	*transport*	SSL protocol type. *0* TLS *1* DTLS
int	*ciphersuites*	Cipher suite.
int	*auth_mode*	Verification method.
int	*sni_enable*	Whether to enable SNI.
char	*ca_cert_path*	CA certificate list and it can be up to *QL_MAX_CA_CERT_CNT*.
char	*own_cert_path*	Local certificate path.
char	*own_key_path*	Local key file path.
char	*own_key_pwd*	The encrypted password for local key file, NULL if there is no encrypted password.

char	*ca_cert_buffer*	Start address of SSL CA certificate buffer array.
int	*ssl_negotiate_timeout*	The maximum timeout of SSL negotiation.
ql_ssl_handshake_timeout_cb	*negotiate_timeout_cb*	SSL shake timeout callback function. See ***Chapter 3.3.3.2*** for details.
void	*negotiate_timeout_cb_arg*	The parameters passed into the SSL handshake timeout callback function.
int	*ignore_invalid_certsign*	Set whether to ignore the check of the SSL certificate sent by the server.
uint32_t	*ignore_certitem*	Set whether to ignore the check of SSL certificate items sent by the server.
int	*ignore_multi_certchain_verify*	Set whether to ignore verification of multi-level certificate chains.
bool	*client_cert_type*	Whether CA certificate saved in file or buffer. *0* Saved in file *1* Saved in buffer
ql_ssl_session	*ssl_session_cache*	SSL session reuse configuration. See ***Chapter 3.3.3.3*** for details.

3.3.3.2. ql_ssl_handshake_timeout_cb

This callback function defines the function pointer of the SSL handshake timeout callback function.

- **Prototype**

```
typedef void(*ql_ssl_handshake_timeout_cb)(ql_ssl_context *ssl_ctx, void *arg)
```

- **Parameter**

ssl_ctx:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

arg:
[In] Pointer to the callback function parameters customized by users.

3.3.3.3. ql_ssl_session

The structure of SSL session reuse configurations:

```
typedef struct {
uint8_t                 session_cache_enable;
ip_addr_t               remote_ip;
uint16_t                remote_port;
uint8_t                 hostname_temp[256];
uint8_t                 session_hostname[256];
mbedtls_ssl_session     ssl_session;
} ql_ssl_session
```

- **Parameter**

Type	Parameter	Description
uint8_t	*session_cache_enable*	Whether to enable session reuse. *0* Disable *1* Enable
ip_addr_t	*remote_ip*	Saves the IP address of the peer server.
uint16_t	*remote_port*	Saves the port number of the peer server.
uint8_t	*hostname_temp*	Stores the host name of the peer server temporarily.
uint8_t	*session_hostname*	Real host name used for session reuse.
mbedtls_ssl_session	*ssl_session*	Stores the current session data.

3.3.4. ql_ssl_conf_set_by_id

This function sets SSL context configuration parameter corresponding to SSL context ID.

- **Prototype**

```
int ql_ssl_conf_set_by_id(int ctx_id, int type, ...)
```

- **Parameter**

ctx_id:
[In] Integer type. SSL context ID. Range: 0–5.

type:
[In] SSL parameter configuration type. See ***Chapter 3.3.2.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.5. ql_ssl_conf_get_by_id

This function gets SSL context configuration parameter corresponding to SSL context ID.

- **Prototype**

```
int ql_ssl_conf_get_by_id(int ctx_id, int type, ...)
```

- **Parameter**

ctx_id:
[In] Integer type. SSL context ID. Range: 0–5.

type:
[In] SSL parameter configuration type. See ***Chapter 3.3.2.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.6. ql_ssl_conf_free

This function releases the resources occupied by SSL configuration parameters.

- **Prototype**

```
int    ql_ssl_conf_free(ql_ssl_config *conf)
```

- **Parameter**

conf:
[In] SSL configuration handle. See ***Chapter 3.3.3.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.7. ql_ssl_new

This function generates a new SSL context.

- **Prototype**

```
int    ql_ssl_new(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[Out] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.7.1. ql_ssl_context

The SSL context is defined as follows:

```
typedef int    ql_ssl_context
```

- **Parameter**

Type	Parameter	Description
int	*ql_ssl_context*	SSL context handle.

3.3.8. ql_ssl_setup

This function sets an SSL context.

- **Prototype**

```
int    ql_ssl_setup(ql_ssl_context *ssl, ql_ssl_config *conf)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

conf:
[In] SSL attribute configuration. See ***Chapter 3.3.3.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.9. ql_ssl_set_socket_fd

This function binds the socket descriptor and SSL context.

- **Prototype**

```
int    ql_ssl_set_socket_fd(ql_ssl_context *ssl, int sock_fd)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

sock_fd:
[In] Socket descriptor.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.10. ql_ssl_set_hostname

This function sets the domain name of the server and is only valid when the SNI is set.

- **Prototype**

```
int    ql_ssl_set_hostname(ql_ssl_context *ssl, const char *hostname)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

hostname:
[In] Server domain name.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.11. ql_ssl_handshake

This function performs the SSL handshake.

- **Prototype**

```
int    ql_ssl_handshark(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.12. ql_ssl_close_notify

This function notifies the server that the SSL connection is closing.

- **Prototype**

```
int    ql_ssl_close_notify(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.13. ql_ssl_get_bytes_avail

This function gets the length of readable data from the data buffer in SSL context after the SSL handshake is completed.

- **Prototype**

```
int    ql_ssl_get_bytes_avail(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

Less than *0*	Returns SSL result codes. See ***Chapter 3.3.1.1*** for details.
Greater than *0*	Returns the length of the readable data in the buffer.

3.3.14. ql_ssl_read

This function reads the readable data from the data buffer in SSL context after the SSL handshake is completed.

- **Prototype**

```
int    ql_ssl_read(ql_ssl_context *ssl, unsigned char *buf, size_t len)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

buf:
[In] The buffer storing readable data.

len:
[In] Length of the data to be read.

- **Return Value**

Less than *0*	Returns SSL result codes. See ***Chapter 3.3.1.1*** for details.
Not less than *0*	Returns the number of data bytes that have been successfully read from the buffer.

3.3.15. ql_ssl_write

This function sends data to server through SSL context after the SSL handshake has finished.

- **Prototype**

```
int    ql_ssl_write(ql_ssl_context *ssl, const unsigned char *buf, size_t len)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

buf:
[In] The buffer for storing data.

len:
[In] Length of the data to be sent.

- **Return Value**

Less than *0*	Returns SSL result codes. See ***Chapter 3.3.1.1*** for details.
Not less than *0*	Returns the number of data bytes that have been successfully read from the buffer.

3.3.16. ql_ssl_free

This function releases the resources occupied by the SSL context.

- **Prototype**

```
int    ql_ssl_free(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

See ***Chapter 3.3.1.1*** for details.

3.3.17. ql_ssl_handshake_finished

This function queries whether the SSL handshake has finished.

- **Prototype**

```
int   ql_ssl_handshark_finished(ql_ssl_context *ssl)
```

- **Parameter**

ssl:
[In] SSL context handle. See ***Chapter 3.3.7.1*** for details.

- **Return Value**

1	The SSL handshake process has completed.
0	The SSL handshake process has not completed.
Less than *0*	Returns SSL result codes. See ***Chapter 3.3.1.1*** for details.

3.3.18. ql_ssl_ciphersuit_is_valid

This function queries whether the specified cipher suite ID is valid.

- **Prototype**

```
int   ql_ssl_ciphersuit_is_valid(int cs_id)
```

- **Parameter**

cs_id:
[In] The cipher suite ID. See *ql_ssl.h* in SDK for details.

- **Return Value**

1	Valid
0	Invalid

4 Example

ssl_demo.c, the example file of SSL API, is located in the *\components\ql-application\ssl* directory of QuecOpen SDK. The example file includes establishing a TCP connection, configuring SSL session attributes, establishing an SSL connection, reading and writing data through the SSL connection and other operations. You can view the complete example of SSL API by yourselves.

5 Appendix References

Table 2: Related Documents

Document Name
[1] Quectel_EC200U_Series_QuecOpen_CSDK_Quick_Start_Guide
[2] Quectel_EC200U_Series_QuecOpen_Log_Capture_Guide
[3] Quectel_EC200U_Series_QuecOpen_Data_Call_API_Reference_Manual

Table 3: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
AP	Application Processor
App	Application
DTLS	Datagram Transport Layer Security
EVB	Evaluation Board
IoT	Internet of Things
PC	Personal Computer
RTOS	Real-Time Operating System
SDK	Software Development Kit
SSL	Secure Sockets Layer
SNI	Server Name Indication
TLS	Transport Layer Security

USB	Universal Serial Bus
UDP	User Datagram Protocol