Quantum Secure Cryptographic library – QSC

Revision 1.0.0.6a, December 11, 2024

John G. Underhill – john.underhill@protonmail.com

This document is an engineering level description of the QSC cryptographic library. In its contents, a guide to implementing QSC, an explanation of its design, as well as references to its component primitives and links to supporting documentation. The QSC library is the property of the author John G. Underhill and the QRCS Corporation.

**Contents Page**

1 [Foreword](#_1:_Foreword) 4

2 [Introduction](#_2:_Introduction) 5

3 [References](#_3:_References) 7

4 [Tables](#_4:_Tables) 8

5 [Symmetric Ciphers](#_5:_Symmetric_Ciphers) 11

5.1 [CSX-512](#CSX512) 11

5.2 [RCS](#RCS) 13

5.3 [ChaCha](#ChaCha) 16

5.4 [AES](#AES) 19

6 [HASH and MAC Functions](#HASH_and_MAC_Functions) 24

6.1 [SHA3](#SHA3) 25

6.2 [SHA2](#SHA2) 29

6.3 [KMAC](#KMAC) 33

6.4 [HMAC](#HMAC) 36

7 [Random Providers](#Random_Providers) 39

7.1 [ACP](#ACP) 39

7.2 [CSP](#CSP) 40

7.3 [RDP](#RDP) 41

8 [DRBG, KDF, and PRNG](#DRBG_KDF_PRNG) 43

8.1 [SHAKE](#SHAKE) 43

8.2 [cSHAKE](#cSHAKE) 45

8.3 [HKDF](#HKDF) 47

8.4 [CSG](#CSG) 48

8.5 [HCG](#HCG) 50

8.6 [SCB](#SCB) 52

8.7 [SECRAND](#SecRand) 54

9 [Asymmetric Ciphers](#Asymmetric_Ciphers) 59

9.1 [ECDH](#ECDH) 59

9.2 [Kyber](#Kyber) 60

9.3 [McEliece](#McEliece) 62

9.4 [NTRU](#NTRU) 64

10 [Asymmetric Signature Schemes](#Asymmetric_Signature_Schemes) 66

10.1 [Dilithium](#Dilithium) 66

10.2 [ECDSA](#ECDSA) 67

10.3 [Falcon](#Falcon) 69

10.4 [SPHINCS+](#SPHINCS) 70

11 [Networking and Sockets](#Networking_and_Sockets) 73

11.1 [IP Info](#IP_Info) 73

11.2 [Queue](#Queue) 77

11.3 [Socket](#Socket) 79

11.4 [Socket Client](#Socket_Client) 89

11.5 [Socket Server](#Socket_Server) 91

11.6 [Network Utilities](#Network_Utilities) 93

12 [Threads and Asynchronous Processing](#Threads_and_Asynchronous_Processing) 98

12.1 [Async Threading](#Asynchronous_Threading) 98

12.2 [Events](#Event) 103

12.3 [Thread Pool](#Thread_Pool) 103

13 [Storage and Data Processing](#Storage_and_Data_Processing) 105

13.1 [Collection](#Collection) 105

13.2 [List](#List) 107

13.3 [QSORT](#QSORT) 110

14 [Integer and String Tools](#Integer_and_String_Tools) 112

14.1 [Array Utilities](#Array_Utilities) 112

14.2 [Console Utilities](#Console_Utilities) 113

14.3 [Encoding](#Encoding) 118

14.4 [File Utilities](#File_Utilities) 119

14.5 [Folder Utilities](#Folder_Utilities) 124

14.6 [Integer Utilities](#Integer_Utilities) 126

14.7 [String Utilities](#String_Utilities) 133

14.8 [System Utilities](#System_Utilities) 140

14.9 [TimerEx](#TimerEx) 143

14.10 [Time Stamp](#Time_Stamp) 144

14.11 [Transpose](#Transpose) 146

14.11 [Winutils](#WinUtils) 147

15 [Memory and Processor](#Memory_and_Processor) 152

15.1 [CPUID](#CPUID) 152

15.2 [SIMD Memory Utilities](#SIMD_Memory_Utilities) 153

15.3 [Secure Memory Functions](#Secure_Memory_Functions) 155

Annex A [Design Considerations](#Design_Decisions) 156

# **1:** **Foreword**

This document is intended as a formal description of the QSC library, an inventory of its contents, function definitions, and brief descriptions of the algorithms. This is a summary of the functions and capabilities of the QSC library version: 1.0.0.6a.

For explanations of the various algorithms implemented in the QSC library, refer to the References section of this document.

QSC has been used to implement several of our products, including the distributed key tunneling protocol SKDP, the multi party crypto-system MPDC, the asymmetric tunneling protocol QSMP, and HKDS, a distributed key management protocol.

# **2:** **Introduction**

QSC is a compact and self-contained post-quantum-secure cryptographic functions library written in the C programming language. It has been written to the highest standards of secure programming, using the MISRA rule-set, and has been designed to be easy to read, verify, and implement. The code in this library is well structured, highly readable and well commented, and thoroughly documented throughout.

It contains only the most secure, proven, and powerful cryptographic primitives. It is focused on the future, not the past, and the future of the worlds secure crypto-systems must be post-quantum secure. We have installed some algorithms for backwards compatibility; AES, ChaCha, SHA2, HMAC, HKDF, ECDH and ECDSA. But our primary design goal has been to create a set of modern tools, ones that can be used to build the security systems of the future.

We have implemented all of the NIST Post Quantum Competition’s winners and several round 3 finalists; the asymmetric ciphers McEliece, Kyber, NTRU, and signature schemes, Dilithium, Falcon, and SPHINCS+. We have formatted them to MISRA secure coding rule-sets, added SIMD instructions using AVX/AVX2, and AVX512 instruction sets.

QSC has a powerful set of tools; an IPv4/IPv6 networking stack featuring synchronous and asynchronous sockets. Multi-threading tools, SIMD memory functions, integer conversion and evaluation functions, as well as a full set of cryptographic hashes, MAC functions, DRBGs, random providers, and random number generators. It is a complete set of tools, all vetted for secure operation, and written to stringent specifications, that can be applied in the most high-security of environments.

QSC also contains two of our symmetric cipher designs, RCS and CSX. These authenticated stream ciphers represent the two cryptographically strongest symmetric ciphers available in the world today, and were designed for true long-term security.

We have implemented the Keccak family of hash functions, and pseudo-random generators including SHA3, SHAKE, and KMAC. Keccak is probably the most well-studied set of cryptographic functions in modern times, having won the NIST SHA3 competition, after being the subject of intense academic scrutiny for more than three years.

The library contains many of the most currently used cryptographic functions, including HMAC, HKDF, SHA2, and AES, providing a window to backwards compatibility in your designs. What we haven’t done, is install anything that we feel is insecure in the current context, while focusing on ciphers and protocols that provide the best guarantee of long-term security. We have made a very modular, compact and efficient library that future security products can be constructed with; our encrypted tunneling protocol QSMP, was built using this library, as well as the key distribution protocols SKDP, and multi-party crypto-system MPDC. Its small size makes it ideal for IOT devices and deployments with memory and storage constraints. QSC can just as easily be used in server/client software, or any implementation that requires a securely coded, fast and powerful set of cryptographic tools. The modular design, makes it ideal as a base tool-set implemented in more complex communications protocols. QSC was designed to be the future of cryptographic development, lean and mean, and containing only the most powerful tools available today.

## **3:** **References**

**3.1 Normative References**

The following documents serve as references for key components in QSC:

1. NIST FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable Output Functions
2. NIST SP 800-185: Derived Functions cSHAKE, KMAC, TupleHash and ParallelHash
3. NIST SP 800-90A: Recommendation for Random Number Generation
4. NIST SP 800-108: Recommendation for Key Derivation using Pseudorandom Functions
5. NIST FIPS 197 The Advanced Encryption Standard
6. FIPS 198.1 The AES standard
7. RFC 2104, HMAC
8. RFC 5869, HKDF

**3.2 Reference Links**

1. The QSC Cryptographic library: <https://github.com/Steppenwolfe65/QSC>
2. The Keccak Code Package: <https://github.com/XKCP/XKCP>
3. NIST AES FIPS 197: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
4. ChaCha Specification: <http://cr.yp.to/chacha/chacha-20080128.pdf>
5. NIST FIPS-202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
6. SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>
7. NIST FIPS 202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
8. NIST Rijndael amended: <http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf>
9. FIPS 198.1 <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>
10. NIST SP800-90B: <http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf>
11. NIST SHA3 The Keccak digest: <http://keccak.noekeon.org/Keccak-submission-3.pdf>
12. ChaCha Specification: <http://cr.yp.to/chacha/chacha-20080128.pdf>
13. RFC 2104, HMAC: <http://tools.ietf.org/html/rfc2104>
14. RFC 5869, HKDF: [http://tools.ietf.org/html/rfc5869](http://tools.ietf.org/html/rfc58694)
15. FIPS 198-1: <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>
16. NaCI library: <https://nacl.cr.yp.to>
17. Kyber a CCA-secure module-lattice-based KEM: <https://eprint.iacr.org/2017/634.pdf>
18. Kyber, a simple, provably secure key exchange: <http://eprint.iacr.org/2012/688.pdf>
19. Classic McEliece: <https://classic.mceliece.org/nist/mceliece-20171129.pdf>
20. The NTRU Algorithm: <https://ntru.org/f/ntru-20190330.pdf>
21. Dilithium Reference: <https://pq-crystals.org/dilithium/data/dilithium-specification.pdf>
22. Dilithium: <https://pq-crystals.org/dilithium/data/dilithium-20180114.pdf>
23. The Falcon Algorithm Specification: <https://falcon-sign.info/falcon.pdf>
24. The SPHINCS+ Signature Scheme: <https://sphincs.org/data/sphincs+-specification.pdf>

# **4:** **Tables**

**Contents Page**

Table 5.1a CSX-512 state structure 11

Table 5.1b CSX-512 keyparams structure 11

Table 5.1c CSX-512 constants 12

Table 5.2a CSX-512 state structure 12

Table 5.2b CSX-512 keyparams structure 13

Table 5.2c RSX constants 13

Table 5.3a ChaCha state structure 16

Table 5.3b ChaCha keyparams structure 16

Table 5.3c ChaCha constants 17

Table 5.4a AES state structure 19

Table 5.4b AES keyparams structure 19

Table 5.4c AES keyparams structure 20

Table 5.4d AES constants 20

Table 6.1a SHA3 state structure 25

Table 6.1b SHA3 constants 26

Table 6.2a SHA2-256 state structure 29

Table 6.2b SHA2-512 state structure 29

Table 6.2c SHA2 constants 29

Table 6.3a Keccak state structure 30

Table 6.4a HMAC-256 state structure 36

Table 6.4b HMAC-512 state structure 36

Table 7.1a ACP constants 39

Table 7.2a CSP constants 40

Table 7.3a RDP constants 41

Table 8.1a Keccak state structure 43

Table 8.4a CSG state structure 43

Table 8.4b CSG constants 48

Table 8.5a HCG structure 50

Table 8.5b HCG constants 50

Table 8.6a secrand state structure 52

Table 8.6b secrand constants 52

Table 9.1a ECDH constants 59

Table 9.2a Kyber constants 60

Table 9.2b Kyber parameter sets 60

Table 9.3a McEliece constants 62

Table 9.3b McEliece parameter sets 62

Table 9.4a NTRU constants 64

Table 9.4b NTRU parameter sets 64

Table 10.1a Dilithium constants 66

Table 10.1b Dilithium parameter sets 66

Table 10.2a ECDSA constants 68

Table 10.3a Falcon constants 69

Table 10.3b Falcon parameter sets 69

Table 10.4a SPHINCS+ constants 71

Table 10.4b SPHINCS+ parameter sets 72

Table 11.1a IPv4 address structure 73

Table 11.1b IPv6 address structure 73

Table 11.1c IPv4 address info structure 73

Table 11.1d IPv6 address info structure 73

Table 11.5 address info constants 73

Table 11.2a queue constants 77

Table 11.3a socket state structure 79

Table 11.3b socket async receive structure 79

Table 11.3c socket receive poll structure 80

Table 11.3d socket state structure 80

Table 11.3e socket state constants 80

Table 11.3f IPv6 prefix types enumeration 81

Table 11.3g address families enumeration 81

Table 11.3h socket states enumeration 81

Table 11.3i socket options enumeration 82

Table 11.3j socket protocols enumeration 82

Table 11.3k socket receive enumeration 83

Table 11.3l socket send enumeration 83

Table 11.3m socket shut down enumeration 83

Table 11.3n socket transports enumeration 83

Table 11.5a socket async receive structure 91

Table 11.5b socket async accept structure 91

Table 11.6a network adaptor info structure 94

Table 11.6b network utilities constants 95

Table 12.1a async type definitions 98

Table 12.2a event types 101

Table 12.2b event list enumeration 102

Table 12.2c event handler structure 102

Table 12.4a thread pool structure 103

Table 13.1a socket state constants 111

Table 13.2a font color enumeration 113

Table 13.2b font style enumeration 113

Table 13.6a common directories enumeration 114

Table 13.7a socket state constants 132

Table 13.18 system constants 140

Table 13.18b drive space structure 140

Table 13.19a timestamp constants 144

Table 14.1 CPU features structure 152

## **5: Symmetric Ciphers**

**5.1** **CSX-512**

**Header:**

csx.h

**Description:**

CSX is based on the ChaCha stream cipher, with some important differences. We doubled the size of the input cipher key to 512-bits. We accomplished this by changing the ciphers transform functions internal integer size from 32-bit integers, to 64-bit integers. This doubled the size of the internal state from 512-bit to 1024-bit. The integer rotation constants were also changed to align with 64-bit integers, and the number of transformation rounds have been increased from the ChaCha maximum of 20 to 40 rounds. CSX is an authenticated AEAD stream-cipher, using the Keccak message authentication code generator KMAC, to authenticate the message stream.

**Structures:**

The **qsc\_csx\_state** structure contains the internal cipher state used by the cipher, along with the Keccak state used by the authentication function KMAC, and a Boolean storing the cipher mode; encrypt or decrypt.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint64 Array | 1024 | Cipher state |
| kstate | Uint64 Array | 1600 | Keccak state |
| encrypt | Bool | 8 | Mode |

Table 5.1a CSX-512 state structure

The **qsc\_csx\_keyparams** structure is used to load the symmetric cipher key, the key length, nonce, and mode into the cipher through the initialization function.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Cipher key |
| keylen | Uint64 | 64 | Key size |
| nonce | Uint8 Array | 128 | Cipher nonce |
| info | Uint8 Array | variable | Optional tweak |
| infolen | Uint64 | 64 | Info size |

Table 5.1b CSX-512 keyparams structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_CSX\_AUTHENTICATED | N/A | Enables KMAC authentication mode. |
| QSC\_CSX\_KPA\_AUTHENTICATION | N/A | Toggles authentication between KMAC and KPA, default is KPA. |
| QSC\_CSX\_BLOCK\_SIZE | 128 | The internal block size in bytes. |
| QSC\_CSX\_INFO\_SIZE | 48 | The maximum byte length of the info string. |
| QSC\_CSX\_KEY\_SIZE | 64 | The size in bytes of the CSX-512 input cipher-key. |
| QSC\_CSX\_MAC\_SIZE | 64 | The CSX-512 MAC code array length in bytes. |
| QSC\_CSX\_NONCE\_SIZE | 16 | The byte size of the nonce array. |
| QSC\_CSX\_STATE\_SIZE | 16 | The uint64 size of the internal state array. |

Table 5.1c CSX-512 constants

**Call Hierarchy:**

The initialize function is called to add the keying material and prepare the CSX state. The associated data call can optionally be used to add data like packet headers, to the authentication code generator’s message input. If the mode is set to encrypt, the transform call encrypts the input message, and outputs the cipher-text and MAC tag. In decrypt mode, the cipher authenticates the cipher-text and if successful, decrypts the cipher-text to the output array. The dispose function resets the cipher state to zero.

initialize(*state*, *key*, *mode*)

associated(*state*, *message*, *length*)

transform(*state*, *output*, *input*, *length*)

dispose(state)

**API:**

**Dispose**

Dispose of the CSX cipher state, erases the state and resets it to zero. The dispose function takes a pointer to the CSX state as a parameter.

void qsc\_csx\_dispose(qsc\_csx\_state\* ctx)

**Initialize**

Initializes the cipher state with the state structure, keyparams structure, and cipher mode. The initialize function takes a pointer to the CSX state, key parameters structure, and a Boolean mode assignment as parameters.

void qsc\_csx\_initialize(qsc\_csx\_state\* ctx, const qsc\_csx\_keyparams\* keyparams, bool encryption)

**Set Associated**

Adds associated data to the authentication mechanism, takes the cipher state, and the data, and data length as parameters. The set associated function takes a pointer to the CSX state, the data array pointer, and the data length as parameters.

void qsc\_csx\_set\_associated(qsc\_csx\_state\* ctx, const uint8\_t\* data, size\_t length)

**Transform**

Encrypts data and adds the authentication code to the cipher-text in encrypt mode, or authenticates the cipher-text and decrypts data in decrypt mode. Takes a pointer to the CSX state structure, a pointer to the output and input arrays, and the data length as parameters.

bool qsc\_csx\_transform(qsc\_csx\_state\* ctx, uint8\_t\* output, const uint8\_t\* input, size\_t length)

**References:**

* ChaCha Specification: <http://cr.yp.to/chacha/chacha-20080128.pdf>
* NIST FIPS-202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>
* NIST FIPS 202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>

**5.2** **RCS**

**Header:**

rcs.h

**Description:**

RCS is an authenticated stream cipher based on the block cipher Rijndael, the symmetric cipher used in AES. We used the wide-block form with a 256-bit internal state, and replaced the differentially weak key-schedule with the Keccak XOF function cSHAKE. We increased the number of transformation rounds from 14 with AES-256, to 22 when using a 256-bit key, or 30 rounds when using a 512-bit key. RCS is an authenticated AEAD stream-cipher, using the Keccak message authentication code generator KMAC, to authenticate the message stream.

**Structures:**

The **qsc\_rcs\_state** structure contains the internal round-key state used by the cipher, along with the Keccak state used by the authentication function KMAC, the nonce array, an internal counter, and a Boolean storing the cipher mode; encrypt or decrypt.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| ctype | Enum | 8 | Key mode |
| roundkeys | Uint32 Array | variable | Key state |
| roundkeylen | Uint64 | 64 | Round-key size |
| rounds | Uint64 | 64 | Round count |
| kstate | Uint64 Array | variable | KMAC state |
| nonce | Uint8 Array | 256 | Nonce array |
| counter | Uint64 | 64 | Internal counter |
| encrypt | Bool | 8 | Mode |

Table 5.2a CSX-512 state structure

The **qsc\_rcs\_keyparams** structure is used to load the symmetric cipher key, the key length, nonce, and mode into the cipher through the initialization function.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Cipher key |
| keylen | Uint64 | 64 | Key size |
| nonce | Uint8 Array | 128 | Cipher nonce |
| info | Uint8 Array | variable | Optional tweak |
| infolen | Uint64 | 64 | Info size |

Table 5.2b CSX-512 keyparams structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_RCS\_AUTHENTICATED | N/A | Enables KMAC authentication mode. |
| QSC\_RCS\_KMACR12 | N/A | Enables the reduced rounds KMAC-R12 implementation. |
| QSC\_RCS\_AESNI\_ENABLED | N/A | Enable the use of intrinsic and the AES-NI implementation. |
| QSC\_ RCS \_BLOCK\_SIZE | 32 | The internal block size in bytes. |
| QSC\_ RCS \_INFO\_SIZE | 48 | The maximum byte length of the info string. |
| QSC\_ RCS256 \_KEY\_SIZE | 32 | The size in bytes of the RCS-256 input cipher-key. |
| QSC\_ RCS256 \_MAC\_SIZE | 32 | The RCS-256 MAC code array length in bytes. |
| QSC\_ RCS512 \_KEY\_SIZE | 64 | The size in bytes of the RCS-512 input cipher-key. |
| QSC\_ RCS512 \_MAC\_SIZE | 64 | The RCS-512 MAC code array length in bytes. |
| QSC\_ RCS \_NONCE\_SIZE | 32 | The byte size of the nonce array. |
| QSC\_ RCS \_STATE\_SIZE | 16 | The uint64 size of the state array. |

Table 5.2c RSX constants

**Call Hierarchy:**

The initialize function is called to add the keying material and prepare the RCS state. The associated data call can optionally be used to add data, like packet headers, to the authentication code generator’s message input. If the mode is set to encrypt, the transform call encrypts the input message, and outputs the cipher-text and MAC tag. In decrypt mode, the RCS authenticates the cipher-text and if successful, decrypts the cipher-text to the output array. The dispose function resets the cipher state to zero.

initialize(*state*, *key*, *mode*)

associated(*state*, *message*, *length*)

transform(*state*, *output*, *input*, *length*)

dispose(*state*)

**API:**

**Dispose**

Dispose of the RCS cipher state, erases the state and resets it to zero. The dispose function takes a pointer to the RCS state as a parameter.

void qsc\_rcs\_dispose(qsc\_rcs\_state\* ctx)

**Initialize**

Initializes the cipher state with the state structure, keyparams structure, and cipher mode. The initialize function takes a pointer to the RCS state, key parameters structure, and a Boolean encryption mode as parameters.

void qsc\_rcs\_initialize(qsc\_rcs\_state\* ctx, const qsc\_rcs\_keyparams\* keyparams, bool encryption)

**Set Associated**

Adds associated data to the authentication mechanism, takes the cipher state, and the data and data length as parameters. The set associated function takes a pointer to the RCS state, the data array pointer, and the data length as parameters.

void qsc\_rcs\_set\_associated(qsc\_rcs\_state\* ctx, const uint8\_t\* data, size\_t length)

**Transform**

Encrypts data and adds the authentication code to the cipher-text in encrypt mode, or authenticates the cipher-text and decrypts data in decrypt mode. Takes a pointer to the RCS state structure, pointers to the output and input arrays, and the data length as parameters.

void qsc\_rcs\_transform(qsc\_rcs\_state\* ctx, uint8\_t\* output, const uint8\_t\* input, size\_t length)

**5.3** **ChaCha**

**Header:**

chacha.h

**Description:**

The ChaCha implementation, like all of our cipher implementations has been optimized with AVX instructions, including AVX2 and AVX-512 implementations. It is the standard implementation of ChaChaPoly20, set to 20 transformation rounds.

**Structures:**

The **qsc\_chacha\_state** structure contains the internal cipher state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Cipher key |
| keylen | Uint64 | 64 | Key size |
| nonce | Uint8 Array | 128 | Cipher nonce |

Table 5.3a ChaCha state structure

The **qsc\_chacha\_keyparams** structure is used to load the symmetric cipher key, the key length, and the nonce into the cipher through the initialization function.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Cipher key |
| keylen | Uint64 | 64 | Key size |
| nonce | Uint8 Array | 64 | Cipher nonce |

Table 5.3b ChaCha keyparams structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_CHACHA\_BLOCK\_SIZE | 64 | The internal block size in bytes. |
| QSC\_CHACHA\_KEY128\_SIZE | 16 | The size in bytes of the CHACHA-128 input cipher-key. |
| QSC\_CHACHA\_KEY256\_SIZE | 32 | The size in bytes of the CHACHA-256 input cipher-key. |
| QSC\_CHACHA\_NONCE\_SIZE | 8 | The CHACHA nonce length in 32-bit integers. |
| QSC\_CHACHA\_ROUND\_COUNT | 20 | The number of transformation rounds. |

Table 5.3c ChaCha constants

**Call Hierarchy:**

The initialize function is called to add the keying material and prepare the ChaCha state. If the mode is set to encrypt, the transform call encrypts the input message, and outputs the cipher-text. In decrypt mode, the cipher decrypts the cipher-text to the output array. The dispose function resets the cipher state to zero.

initialize(*state*, *key*, *mode*)

transform(*state*, *output*, *input*, *length*)

dispose(*state*)

**API:**

**Dispose**

Dispose of the ChaCha cipher state, erases the state and resets it to zero. The dispose function takes a pointer to the ChaCha state as a parameter.

void qsc\_chacha\_dispose(qsc\_chacha\_state\* ctx)

**Initialize**

Initializes the cipher state with the state structure, keyparams structure, and cipher mode. The initialize function takes a pointer to the ChaCha state, and a pointer to the key parameters structure as parameters.

void qsc\_chacha\_initialize(qsc\_chacha\_state\* ctx, const qsc\_chacha\_keyparams\* keyparams)

**Transform**

Encrypts data from the input array, and writes the cipher-text to the output array in encrypt mode. In decrypt mode, input the cipher-text, and outputs the plain-text. Takes a pointer to the ChaCha state structure, a pointer to the output and input arrays, and the input length as parameters.

void qsc\_chacha\_transform(qsc\_chacha\_state\* ctx, uint8\_t\* output, const uint8\_t\* input, size\_t length)

**References:**

* ChaCha Specification: <http://cr.yp.to/chacha/chacha-20080128.pdf>

**5.4** **AES**

**Header:**

aes.h

**Description:**

The Advanced Encryption Standard (AES) implementation of the Rijndael cipher. Includes ECB mode for testing, as well as the CBC and CTR cipher modes, and the authenticated stream cipher mode HBA. Implementations of AES-NI and the reference code versions, enhanced with AVX to AVX-512 instructions.

**Structures:**

The **qsc\_aes\_state** structure contains the internal cipher state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| roundkeys | Uint32 Array | variable | Key state |
| roundkeylen | Uint64 | 64 | Round-key size |
| rounds | Uint64 | 64 | Round count |
| nonce | Uint8 Array | 256 | Nonce array |

Table 5.4a AES state structure

The **qsc\_aes\_keyparams** structure is used to load the symmetric cipher key, the key length, and the nonce into the cipher through the initialization function.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Cipher key |
| keylen | Uint64 | 64 | Key size |
| iv | Uint8 Array | 64 | Cipher IV |

Table 5.4b AES keyparams structure

The **qsc\_aes\_hba256\_state** structure is used with the HBA authenticated AEAD mode.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| kstate | MAC State | Variable | HMAC or KMAC state |
| cstate | AES State | Variable | The AES state |
| mkey | Uint8 Array | 256 | Mac Key |
| cust | Uint8 Array | 2048 | Custom string |
| custlen | Uint64 | 64 | Custom string length |
| encrypt | Boolean | 8 | Encrypt/decrypt mode |

Table 5.4c AES keyparams structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SYSTEM\_AESNI\_ENABLED | N/A | Enable the use of intrinsics and the AES-NI implementation. |
| QSC\_AES\_BLOCK\_SIZE | 16 | The internal block size in bytes, required by the encryption and decryption functions. |
| QSC\_AES128\_KEY\_SIZE | 16 | The size in bytes of the AES-128 input cipher-key. |
| QSC\_AES256\_KEY\_SIZE | 32 | The size in bytes of the AES-256 input cipher-key. |
| QSC\_AES\_IV\_SIZE | 32 | The AES initialization vector byte length. |
| QSC\_HBA256\_MAC\_LENGTH | 20 | The HBA modes maximum info string length. |
| QSC\_HBA\_MAXAAD\_SIZE | 256 | The HBA modes maximum AAD string length. |
| QSC\_HBA\_MAXINFO\_SIZE | 256 | The HBA modes maximum info string length. |
| QSC\_HBA\_KMAC\_EXTENSION | N/A | Enables the HBA KMAC authentication mode. |
| QSC\_HBA\_HMAC\_EXTENSION | N/A | Enables the HBA HMAC authentication mode. |

Table 5.4d AES constants

**Call Hierarchy:**

The initialize function is called to add the keying material and prepare the AES state. If the mode is set to encrypt, the transform call encrypts the input message, and outputs the cipher-text. In decrypt mode, the cipher decrypts the cipher-text to the output array. The dispose function resets the cipher state to zero.

initialize(*state*, *key*, *mode*, *type*)

decrypt(*state*, *output*, *input*, *length*)

encrypt(*state*, *output*, *input*, *length*)

dispose(*state*)

**API:**

**Dispose**

Dispose of the AES cipher state, erases the state and resets it to zero. The dispose function takes a pointer to the AES state as a parameter.

void qsc\_aes\_dispose(qsc\_aes\_state\* state)

**Initialize**

Initializes the cipher state with the state structure, keyparams structure, and cipher mode. The initialize function takes a pointer to the AES state, key parameters structure, and a Boolean encryption mode as parameters.

void qsc\_aes\_initialize(qsc\_aes\_state\* state, const qsc\_aes\_keyparams\* keyparams, bool encryption, qsc\_aes\_cipher\_type ctype)

**CBC Encrypt**

Encrypts data from the input array, and writes the cipher-text to the output array in encrypt mode. Takes a pointer to the AES state, pointers to the input and output arrays, and input length as parameters.

void qsc\_aes\_cbc\_encrypt(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input, size\_t inputlen)

**CBC Decrypt**

Decrypts cipher-text from the input array, and writes the plain-text to the output array in decrypt mode. Takes a pointer to the AES state, pointers to the input and output arrays, and length as parameters.

void qsc\_aes\_cbc\_decrypt(qsc\_aes\_state\* state, uint8\_t\* output, size\_t \*outputlen, const uint8\_t\* input, size\_t inputlen)

**CBC Encrypt Block**

Encrypt a single block of data using cipher block chaining mode. Takes a pointer to the AES state, pointers to the input and output arrays as parameters.

void qsc\_aes\_cbc\_encrypt\_block(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input)

**CBC Decrypt Block**

Decrypt a single block of data using cipher block chaining mode. Takes a pointer to the AES state, pointers to the input and output arrays as parameters.

void qsc\_aes\_cbc\_decrypt\_block(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input)

**Add Padding**

Add PKCS7 padding to a CBC block before encryption. The function takes the input array pointer and array length as parameters.

void qsc\_pkcs7\_add\_padding(uint8\_t\* input, size\_t length)

**Padding Size**

Calculates the padding length on a CBC block after decryption. The function takes the input array pointer as a parameter, and returns the padding size.

size\_t qsc\_pkcs7\_padding\_length(const uint8\_t\* input)

**CTR-BE Transform**

Transform data using AES with a Big-Endian counter mode. This function takes a pointer to the AES state, pointers to the input and output arrays, and the input length as parameters.

void qsc\_aes\_ctrbe\_transform(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input, size\_t inputlen)

**CTR-LE Transform**

Transform data using AES with a Little-Endian counter mode. This function takes a pointer to the AES state, pointers to the input and output arrays, and the length as parameters.

void qsc\_aes\_ctrle\_transform(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input, size\_t inputlen)

**ECB Encrypt Block**

Used for encryption in known answer tests or as a component in a more complex protocol. This function takes a pointer to the AES state, pointers to the input and output arrays as parameters.

void qsc\_aes\_ecb\_decrypt\_block(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input)

**ECB Decrypt Block**

Used for decryption in known answer tests or as a component in a more complex protocol. This function takes a pointer to the AES state, pointers to the input and output arrays as parameters.

void qsc\_aes\_ecb\_encrypt\_block(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input)

**HBA**

The HBA cipher mode, turns AES into an authenticated AEAD stream cipher, using either HMAC or KMAC message authentication code generator (MAC) functions.

**Dispose**

Clears the HBA state, resetting the state members to their defaults. Takes a pointer to the AES state as a parameter.

void qsc\_aes\_hba256\_dispose(qsc\_aes\_hba256\_state\* state)

**Initialize**

Initialize the HBA mode with a keyparams structure. Takes a pointer to the HBA state, the keyparams with a key, nonce, and optional info arrays, and the encryption mode as function parameters.

void qsc\_aes\_hba256\_initialize(qsc\_aes\_hba256\_state\* state, const qsc\_aes\_keyparams\* keyparams, bool encrypt)

**Set Associated**

The set associated function adds additional message data to the MAC function. Takes a pointer to the HBA state, a pointer to the message data, and the message length as parameters.

void qsc\_aes\_hba256\_set\_associated(qsc\_aes\_hba256\_state\* state, const uint8\_t\* data, size\_t datalen)

**Transform**

The transform function encrypts and MACS the cipher-text in encryption mode, or authenticates the cipher-text and decrypts the message in decryption mode. The function takes a pointer to the HBA state, pointers to the output and input arrays, and the message length as parameters.

bool qsc\_aes\_hba256\_transform(qsc\_aes\_hba256\_state\* state, uint8\_t\* output, const uint8\_t\* input, size\_t inputlen)

**References:**

* NIST AES FIPS 197: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
* NIST Rijndael amended: <http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf>

## **6:** **HASH and MAC Functions**

**6.1** **SHA3**

**Header:**

sha3.h

**Description:**

The SHA3 cryptographic message digest, the NIST standard in secure hash functions and supporting protocols, is based on the Keccak family of cryptographic primitives. The SHA3 hash comes in three variants; SHA3-128, SHA3-256, and SHA3-512. The 128-bit variant outputs 16 bytes of hash code, the 256-bit variant outputs a 32-byte hash code, while the 512-bit variant produces 64 bytes of code.

**Structures:**

The **qsc\_keccak\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint64 Array | 1600 | Hash state |
| buffer | Uint8 Array | 1600 | Message buffer |
| position | Uint64 | 64 | Buffer position |

Table 6.1a SHA3 state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_KECCAK\_CSHAKE\_DOMAIN\_ID | 0x04 | The cSHAKE domain id |
| QSC\_KECCAK\_KMAC\_DOMAIN\_ID | 0x10 | The KMAC domain id. |
| QSC\_KECCAK\_KPA\_DOMAIN\_ID | 0x41 | The KPA domain id. |
| QSC\_KECCAK\_PERMUTATION\_ROUNDS | 24 | The standard number of permutation rounds. |
| QSC\_KECCAK\_PERMUTATION\_MAX\_ROUNDS | 48 | The maximum number of permutation rounds. |
| QSC\_KECCAK\_PERMUTATION\_MIN\_ROUNDS | 12 | The minimum number of permutation rounds. |
| QSC\_KECCAK\_SHA3\_DOMAIN\_ID | 6 | The SHA3 domain id. |
| QSC\_KECCAK\_SHAKE\_DOMAIN\_ID | 0x1F | The SHAKE domain id. |
| QSC\_KECCAK\_STATE\_BYTE\_SIZE | 200 | The Keccak state array byte size. |
| QSC\_KECCAK\_128\_RATE | 168 | The KMAC-128 byte absorption rate. |
| QSC\_KECCAK\_256\_RATE | 136 | The KMAC-256 byte absorption rate. |
| QSC\_KECCAK\_512\_RATE | 72 | The KMAC-512 byte absorption rate. |
| QSC\_KECCAK\_STATE\_SIZE | 25 | The Keccak SHA3 uint64 state array size. |
| QSC\_KMAC\_256\_KEY\_SIZE | 32 | The KMAC-256 key size in bytes. |
| QSC\_KMAC\_512\_KEY\_SIZE | 64 | The KMAC-512 key size in bytes. |
| QSC\_SHA3\_128\_HASH\_SIZE | 16 | The SHA3-128 hash size in bytes. |
| QSC\_SHA3\_256\_HASH\_SIZE | 32 | The SHA-256 hash size in bytes. |
| QSC\_SHA3\_512\_HASH\_SIZE | 64 | The SHA-512 hash size in bytes. |
| QSC\_SHAKE\_256\_KEY\_SIZE | 32 | The SHAKE-256 key size in bytes. |
| QSC\_SHAKE512\_KEY\_SIZE | 64 | The SHAKE-512 key size in bytes. |

Table 6.1b SHA3 constants

**Call Hierarchy**

The long form of the SHA3 function API, first requires a call to the initialize function to ready the SHA3 state, then calls to the update function to add message data to the hash. The finalize call writes the message hash to the output array, and the dispose function sets the state to zero.

initialize(*state*)

update(*state*, *message*, *length*)

finalize(*state*, *output*)

dispose(*state*)

**API:**

**Compute**

Compute the hash for a message with a single function call. Takes the message, and hashes it, and returns the hash code to the output array. Takes pointers to the output and message arrays, and the message length as parameters.

void qsc\_sha3\_compute128(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

void qsc\_sha3\_compute256(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

void qsc\_sha3\_compute512(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

**Dispose**

Clear the Keccak state structure, setting all variables to zero. Takes a pointer to the Keccak state as a parameter.

void qsc\_keccak\_dispose(qsc\_keccak\_state\* ctx)

**Finalize**

Process the SHA3 state and return the output hash. Takes a pointer to the Keccak state, the permutation rate, the domain separator, and a pointer to the output array as parameters.

void qsc\_keccak\_finalize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t outlen, uint8\_t domain, size\_t rounds)

**Initialize**

Prepares the SHA3 state structure, must be called before any other call. Takes the Keccak state as a parameter.

void qsc\_sha3\_initialize(qsc\_keccak\_state\* ctx)

**Update**

Update the SHA3 state with message data. Takes a pointer to the state, the permutation rate, a pointer to the message array, and the message length as parameters.

void qsc\_sha3\_update(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* message, size\_t msglen)

**Permute**

The SHA3 permutation functions are exposed, for use as a function in other protocols. Takes a pointer to the Keccak state, and a round count as parameters.

void qsc\_keccak\_permute(qsc\_keccak\_state\* ctx, size\_t rounds)

void qsc\_keccak\_permute\_p1600c(uint64\_t\* state, size\_t rounds)

void qsc\_keccak\_permute\_p1600u(uint64\_t\* state)

**Keccak**

Keccak free functions used in the hash, XOF and MAC functions, have been exposed, so that they can be used in more complex constructions.

**API:**

void qsc\_keccak\_absorb(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* message, size\_t msglen, uint8\_t domain, size\_t rounds)

void qsc\_keccak\_absorb\_custom(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* custom, size\_t custlen, const uint8\_t\* name, size\_t namelen, size\_t rounds)

void qsc\_keccak\_absorb\_key\_custom(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen, const uint8\_t\* name, size\_t namelen, size\_t rounds)

void qsc\_keccak\_finalize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t outlen, uint8\_t domain, size\_t rounds)

void qsc\_keccak\_incremental\_absorb(qsc\_keccak\_state\* ctx, uint32\_t rate, const uint8\_t\* message, size\_t msglen)

void qsc\_keccak\_incremental\_finalize(qsc\_keccak\_state\* ctx, uint32\_t rate, uint8\_t domain)

void qsc\_keccak\_incremental\_squeeze(qsc\_keccak\_state\* ctx, size\_t rate, uint8\_t\* output, size\_t outlen)

void qsc\_keccak\_permute(qsc\_keccak\_state\* ctx, size\_t rounds)

void qsc\_keccak\_permute\_p1600c(uint64\_t\* state, size\_t rounds)

void qsc\_keccak\_squeezeblocks(qsc\_keccak\_state\* ctx, uint8\_t\* output, size\_t nblocks, qsc\_keccak\_rate rate, size\_t rounds)

void qsc\_keccak\_initialize\_state(qsc\_keccak\_state\* ctx)

void qsc\_keccak\_update(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* message, size\_t msglen, size\_t rounds)

**KPA**

The Keccak Parallel Authentication functions, process a message in parallel using SIMD instructions.

**API:**

void qsc\_kpa\_dispose(qsc\_kpa\_state\* ctx)

void qsc\_kpa\_finalize(qsc\_kpa\_state\* ctx, uint8\_t\* output, size\_t outlen)

void qsc\_kpa\_initialize(qsc\_kpa\_state\* ctx, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen)

void qsc\_kpa\_update(qsc\_kpa\_state\* ctx, const uint8\_t\* message, size\_t msglen)

**References:**

* SHA3 Fips202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>
* SHA3 Keccak Submission <http://keccak.noekeon.org/Keccak-submission-3.pdf>
* Keccak Reference Guide: <https://keccak.team/files/Keccak-reference-3.0.pdf>

**6.2** **SHA2**

**Header:**

sha2.h

**Description:**

The SHA2 cryptographic message digest, is the NIST standard in secure hash functions and supporting protocols. The SHA2 hash comes in two variants; SHA2-256, and SHA2-512. The 256-bit variant outputs a 32-byte hash code, while the 512-bit variant produces 64 bytes of code.

**Structures:**

The **qsc\_sha256\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint32 Array | 256 | Hash state |
| buffer | Uint8 Array | 512 | Message buffer |
| t | Uint64 | 64 | Message length |
| position | Uint64 | 64 | Buffer position |

Table 5.2a SHA2-256 state structure

The **qsc\_sha512\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint64 Array | 512 | Hash state |
| buffer | Uint8 Array | 512 | Message buffer |
| t | Uint64 Array | 128 | Message length |
| position | Uint64 | 64 | Buffer position |

Table 6.2b SHA2-512 state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SHA2\_SHANI\_ENABLED | N/A | Enables the SHA2 permutation intrinsics. |
| QSC\_HKDF\_256\_KEY\_SIZE | 32 | The HKDF-256 key size in bytes. |
| QSC\_HKDF\_512\_KEY\_SIZE | 64 | The HKDF-512 key size in bytes. |
| QSC\_HMAC\_256\_KEY\_SIZE | 32 | The HMAC(SHA2-256) key size. |
| QSC\_HMAC\_512\_KEY\_SIZE | 64 | The HMAC(SHA2-512) key size. |
| QSC\_HMAC\_256\_MAC\_SIZE | 32 | The HMAC-256 mac-code size in bytes. |
| QSC\_HMAC\_512\_MAC\_SIZE | 64 | The HMAC-512 mac-code size in bytes. |
| QSC\_HMAC\_256\_RATE | 64 | The HMAC-256 input rate size in bytes. |
| QSC\_HMAC\_512\_RATE | 128 | The HMAC-512 input rate size in bytes. |
| QSC\_SHA2\_256\_HASH\_SIZE | 32 | The SHA2-256 hash size in bytes. |
| QSC\_SHA2\_384\_HASH\_SIZE | 48 | The SHA2-384 hash size in bytes. |
| QSC\_SHA2\_512\_HASH\_SIZE | 64 | The SHA2-512 hash size in bytes. |
| QSC\_SHA2\_256\_RATE | 64 | The SHA-256 byte absorption rate. |
| QSC\_SHA2\_384\_RATE | 128 | The SHA-384 byte absorption rate. |
| QSC\_SHA2\_512\_RATE | 128 | The SHA2-512 byte absorption rate. |
| QSC\_SHA2\_STATE\_SIZE | 0x08 | The SHA2-256 state array size. |

Table 6.2c SHA2 constants

**Call Hierarchy**

The long form of the SHA2 function API, first requires a call to the initialize function to ready the SHA2 state, then calls to the update function to add message data to the hash. The finalize call writes the message hash to the output array, and the dispose function sets the state to zero.

initialize(*state*)

update(*state*, *message*, *length*)

finalize(*state*, *output*)

dispose(*state*)

**API:**

**Compute**

Compute the hash for a message with a single function call. Takes the message, and hashes it, and returns the hash code to the output array. This function takes pointers to the output and message arrays, and the message length as parameters.

void qsc\_sha256\_compute(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

void qsc\_sha384\_compute(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

void qsc\_sha512\_compute(uint8\_t\* output, const uint8\_t\* message, size\_t msglen)

**Dispose**

Clear the SHA2 state structure, setting all variables to zero. Takes a pointer to the SHA2 state as a parameter.

void qsc\_sha256\_dispose(qsc\_sha256\_state\* ctx)

void qsc\_sha384\_dispose(qsc\_sha384\_state\* ctx)

void qsc\_sha512\_dispose(qsc\_sha512\_state\* ctx)

**Finalize**

Process the SHA2 state and return the output hash. Takes a pointer to the SHA2 state, and a pointer to the output array as parameters.

void qsc\_sha256\_finalize(qsc\_sha256\_state\* ctx, uint8\_t\* output)

void qsc\_sha384\_finalize(qsc\_sha384\_state\* ctx, uint8\_t\* output)

void qsc\_sha512\_finalize(qsc\_sha512\_state\* ctx, uint8\_t\* output)

**Initialize**

Prepares the SHA2 state structure, must be called before any other call. Takes a pointer to the SHA2 state as a parameter.

void qsc\_sha256\_initialize(qsc\_sha256\_state\* ctx)

void qsc\_sha384\_initialize(qsc\_sha384\_state\* ctx)

void qsc\_sha512\_initialize(qsc\_sha512\_state\* ctx)

**Update**

Update the SHA2 state with message data. Takes a pointer to the SHA2 state, a pointer to the message array, and the message length as parameters.

void qsc\_sha256\_update(qsc\_sha256\_state\* ctx, const uint8\_t\* message, size\_t msglen)

void qsc\_sha384\_update(qsc\_sha384\_state\* ctx, const uint8\_t\* message, size\_t msglen)

void qsc\_sha512\_update(qsc\_sha512\_state\* ctx, const uint8\_t\* message, size\_t msglen)

**Permute**

The SHA2-256 and SHA2-512 permutation functions are exposed, for use as a function in other protocols. Takes a pointer to the output and the input arrays as parameters.

void qsc\_sha256\_permute(uint32\_t\* output, const uint8\_t\* input)

void qsc\_sha512\_permute(uint64\_t\* output, const uint8\_t\* input)

**References:**

* SHA3 Fips202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* SHA3 Keccak Submission: <http://keccak.noekeon.org/Keccak-submission-3.pdf>

**6.3** **KMAC**

**Header:**

sha3.h

**Description:**

The Keccak Message Authentication Code generator (KMAC), is a keyed hash function, used to authenticate a message. It is part of the SHA3 family of functions.

**Structures:**

The KMAC **qsc\_keccak\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint64 Array | 1600 | Hash state |
| buffer | Uint8 Array | 1600 | Message buffer |
| position | Uint64 | 64 | Buffer position |

Table 6.3a Keccak state structure

**Call Hierarchy:**

The long form of the KMAC function API, first requires a call to the initialize function to add the keying material and update the state, then calls to the update function to add message data to the hash. The finalize call writes the message hash to the output array, and the dispose function sets the state to zero.

initialize(*state*, *rate*, *key*, *custom*)

update(*state*, *rate*, *message*, *length*)

finalize(*state*, *rate*, *output*)

dispose(*state*)

**API:**

**Compute**

The compute function adds pointers to the key, customization string, and message, and generates the MAC code with a single function call. Takes pointers to the output, custom, and message arrays, and their lengths as parameters.

void qsc\_kmac128\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen)

void qsc\_kmac256\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen)

void qsc\_kmac512\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen)

**Dispose**

The dispose function resets the MAC functions state to zero. Takes a pointer to the Keccak state as a parameter.

void qsc\_keccak\_dispose(qsc\_keccak\_state\* ctx)

**Finalize**

The finalize function processes the state, and outputs the MAC code. The function takes a pointer to the state, the permutation rate, the output array pointer, and output length as parameters.

void qsc\_kmac\_finalize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t outlen)

**Initialize**

The initialize function initializes the state and adds the key, permutation rate, and customization string. The function takes a pointer the state, the permutation rate, pointers to the key and custom arrays, and their lengths as parameters.

void qsc\_kmac\_initialize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* key, size\_t keylen, const uint8\_t\* custom, size\_t custlen)

**Update**

The update function adds message data to the MAC function. The function takes a pointer the state, the permutation rate, a pointer to the output array, and the output length as parameters.

void qsc\_kmac\_finalize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t outlen)

**Parallel KMAC**

The parallel forms of KMAC process multiple message streams concurrently using SIMD instructions. Takes pointers to multiple output, key, custom and message arrays, and their sizes, as parameters.

void kmac128x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, size\_t msglen)

void kmac256x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, size\_t msglen)

void kmac512x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, size\_t msglen)

void kmac128x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3,

const uint8\_t\* key4, const uint8\_t\* key5, const uint8\_t\* key6, const uint8\_t\* key7, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, const uint8\_t\* cst4, const uint8\_t\* cst5, const uint8\_t\* cst6, const uint8\_t\* cst7, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, const uint8\_t\* msg4, const uint8\_t\* msg5, const uint8\_t\* msg6, const uint8\_t\* msg7, size\_t msglen)

void kmac256x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3,

const uint8\_t\* key4, const uint8\_t\* key5, const uint8\_t\* key6, const uint8\_t\* key7, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, const uint8\_t\* cst4, const uint8\_t\* cst5, const uint8\_t\* cst6, const uint8\_t\* cst7, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, const uint8\_t\* msg4, const uint8\_t\* msg5, const uint8\_t\* msg6, const uint8\_t\* msg7, size\_t msglen)

void kmac512x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* key0, const uint8\_t\* key1, const uint8\_t\* key2, const uint8\_t\* key3,

const uint8\_t\* key4, const uint8\_t\* key5, const uint8\_t\* key6, const uint8\_t\* key7, size\_t keylen, const uint8\_t\* cst0, const uint8\_t\* cst1, const uint8\_t\* cst2, const uint8\_t\* cst3, const uint8\_t\* cst4, const uint8\_t\* cst5, const uint8\_t\* cst6, const uint8\_t\* cst7, size\_t cstlen, const uint8\_t\* msg0, const uint8\_t\* msg1, const uint8\_t\* msg2, const uint8\_t\* msg3, const uint8\_t\* msg4, const uint8\_t\* msg5, const uint8\_t\* msg6, const uint8\_t\* msg7, size\_t msglen)

**References:**

* NIST FIPS-202: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>

**6.4** **HMAC**

**Header:**

sha2.h

**Description:**

The Hash-based Message Authentication Code generator (HMAC), is a keyed hash function, used to authenticate a message. It uses the SHA2 family of hash functions, SHA2-256, and SHA2-512.

**Structures:**

The **qsc\_hmac256\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| pstate | SHA2-256 State | 996 | Hash state |
| ipad | Uint8 Array | 512 | Input buffer |
| opad | Uint8 Array | 512 | Output buffer |

Table 6.4a HMAC-256 state structure

The **qsc\_hmac512\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| pstate | SHA2-512 State | 1216 | Hash state |
| ipad | Uint8 Array | 1024 | Input buffer |
| opad | Uint8 Array | 1024 | Output buffer |

Table 6.4b HMAC-512 state structure

**Call Hierarchy**

The long form of the HMAC function API, first requires a call to the initialize function to add the keying material and initialize the state, then calls to the update function to add message data to the hash. The finalize call writes the message hash to the output array, and the dispose function sets the state to zero.

initialize(*state*, *key*)

update(*state*, *message*, *length*)

finalize(*state*, *output*)

dispose(*state*)

**API:**

**Compute**

The compute function adds the key and message, and generates the MAC code with a single function call. Takes pointers to the output, message, and key arrays, and their lengths as parameters.

void qsc\_hmac256\_compute(uint8\_t\* output, const uint8\_t\* message, size\_t msglen, const uint8\_t\* key, size\_t keylen)

void qsc\_hmac512\_compute(uint8\_t\* output, const uint8\_t\* message, size\_t msglen, const uint8\_t\* key, size\_t keylen)

**Dispose**

The dispose function resets the MAC functions state to zero. Takes a pointer to the HMAC state as a parameter.

void qsc\_hmac256\_dispose(qsc\_hmac256\_state\* ctx)

void qsc\_hmac512\_dispose(qsc\_hmac512\_state\* ctx)

**Finalize**

The finalize function finalizes the state, and outputs the MAC code. Takes pointers to the HMAC state, and the output array as parameters.

void qsc\_hmac256\_finalize(qsc\_hmac256\_state\* ctx, uint8\_t\* output)

void qsc\_hmac512\_finalize(qsc\_hmac512\_state\* ctx, uint8\_t\* output)

**Initialize**

The initialize function initializes the state and adds the key. Takes pointers to the HMAC state, the key array, and key length as parameters.

void qsc\_hmac256\_initialize(qsc\_hmac256\_state\* ctx, const uint8\_t\* key, size\_t keylen)

void qsc\_hmac512\_initialize(qsc\_hmac512\_state\* ctx, const uint8\_t\* key, size\_t keylen)

**Update**

The update function adds message data to the MAC function. Takes pointers to the HMAC state, the message array, and message length as parameters.

void qsc\_hmac256\_update(qsc\_hmac256\_state\* ctx, const uint8\_t\* message, size\_t msglen)

void qsc\_hmac512\_update(qsc\_hmac512\_state\* ctx, const uint8\_t\* message, size\_t msglen)

**References:**

* RFC 2104, HMAC: <http://tools.ietf.org/html/rfc2104>
* FIPS 198-1: <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>

## **7:** **Random Providers**

**7.1** **ACP**

**Header:**

acp.h

**Description:**

The auto entropy provider uses a hashed collection of system timers, statistics,

the RDRAND provider, and the system random provider, to seed an instance of cSHAKE-512.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_ACP\_SEED\_MAX | 1024000 | The maximum seed size that can be extracted from a single generate call. |

Table 7.1a ACP constants

**API:**

**Generate**

Fills an output array with random bytes. Takes a pointer to the output array, and the output length as parameters.

bool qsc\_acp\_generate(uint8\_t\* output, size\_t length)

**ACP Uint16**

Returns a random 16-bit integer.

uint16\_t qsc\_acp\_uint16()

**ACP Uint32**

Returns a random 32-bit integer.

uint16\_t qsc\_acp\_uint32()

**ACP Uint64**

Returns a random 64-bit integer.

uint16\_t qsc\_acp\_uint64()

**7.2** **CSP**

**Header:**

csp.h

**Description:**

The cryptographic systems entropy provider uses entropy provided by the operating systems random provider.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_CSP\_SEED\_MAX | 1024000 | The maximum seed size that can be extracted from a single generate call. |

Table 7.2a CSP constants

**API:**

**Generate**

Fills an output array with random bytes. Takes a pointer to the output array, and the output length as parameters.

bool qsc\_csp\_generate(uint8\_t\* output, size\_t length)

**CSP Uint16**

Returns a random 16-bit integer.

uint16\_t qsc\_csp\_uint16()

**CSP Uint32**

Returns a random 32-bit integer.

uint16\_t qsc\_csp\_uint32()

**CSP Uint64**

Returns a random 64-bit integer.

uint16\_t qsc\_csp\_uint64()

**7.3** **RDP**

**Header:**

rdp.h

**Description:**

The RDRAND entropy provider uses random noise generated by the CPU as a source of random bytes.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_RDP\_SEED\_MAX | 1024000 | The maximum seed size that can be extracted from a single generate call. |

Table 7.3a RDP constants

**API:**

**Generate**

Fills an output array with random bytes. Takes the pointer to the output array, and the output length as parameters.

bool qsc\_rdp\_generate(uint8\_t\* output, size\_t length)

**RDP Uint16**

Returns a random 16-bit integer.

uint16\_t qsc\_rdp\_uint16()

**RDP Uint32**

Returns a random 32-bit integer.

uint16\_t qsc\_rdp\_uint32()

**RDP Uint64**

Returns a random 64-bit integer.

uint16\_t qsc\_rdp\_uint64()

## **8:** **DRBG, KDF, and PRNG**

**8.1** **SHAKE**

**Header:**

sha3.h

**Description:**

SHAKE is the Keccak extended output function (XOF), a secure way to generate pseudo-random output from an input key. It can be used as a key expansion function, a pseudo-random byte generator, a hash function, and as a PRNG.

cSHAKE is the customized from of SHAKE, it takes a key, and the additional customization and name parameters.

**Structures:**

The SHAKE **qsc\_keccak\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| state | Uint64 Array | 1600 | Hash state |
| buffer | Uint8 Array | 1600 | Message buffer |
| position | Uint64 | 64 | Buffer position |

Table 8.1a Keccak state structure

**Call Hierarchy:**

The long form of the SHAKE function API, first requires a call to the initialize function to add the keying material. The squeeze call writes the pseudo-random to the output array, and the dispose function sets the state to zero.

initialize(*state*, *rate*, *key*, *keylen*)

squeeze (*state*, *rate*, *output*, *nblocks*)

dispose(*state*)

**API:**

**Compute**

The compute function is the short form of the API, the key is added and the output is processed using a single function call. The SHAKE compute functions take a pointer to the output array, the number of bytes to write to that array, a pointer to the key array, and the keys byte length as parameters.

void qsc\_shake128\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen)

void qsc\_shake256\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen)

void qsc\_shake512\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen)

**Initialize**

The initialize function is part of the long-form of the API. It is used to initialize the Keccak state, and inject the key. The SHAKE initialize function takes a pointer to the Keccak state, the absorption rate, a pointer to the input key, and the key length as parameters.

void qsc\_shake\_initialize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* key, size\_t keylen)

**Squeeze**

The squeeze function outputs pseudo-random in blocks. The block size corresponds to the Keccak absorption/permutation rate. Initialize must be called first, and the output array must be sized to receive a rate-sized block of bytes. The squeeze function takes a pointer to the Keccak state, the absorption rate, a pointer to the output byte array, and the number of blocks to write as parameters.

void qsc\_shake\_squeezeblocks(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t nblocks)

**Parallel SHAKE**

The parallel SHAKE functions process multiple blocks of data concurrently using SIMD instructions. Using AVX2 instructions, 4 simultaneous SHAKE calculations can be made, the ‘x4’ API. With AVX-512 instructions, 8 simultaneous SHAKE calculations are made, using the ‘x8’ API.

**SHAKE-X4**

The SHAKE X4 functions take 4 output array pointers, and 4 input array pointers, and the input and output arrays lengths as parameters. The functions take multiple output and input array pointers, and the input and output array sizes as parameters.

void shake128x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

size\_t outlen, const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2,

const uint8\_t\* inp3, size\_t inlen)

void shake256x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

size\_t outlen, const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2,

const uint8\_t\* inp3, size\_t inlen)

void shake512x4(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

size\_t outlen, const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2,

const uint8\_t\* inp3, size\_t inlen)

**SHAKE-X8**

The SHAKE X8 functions take 8 output array pointers, and 8 input array pointers, and the input and output arrays lengths as parameters.

void shake128x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2, const uint8\_t\* inp3,

const uint8\_t\* inp4, const uint8\_t\* inp5, const uint8\_t\* inp6, const uint8\_t\* inp7, size\_t inlen)

void shake256x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2, const uint8\_t\* inp3,

const uint8\_t\* inp4, const uint8\_t\* inp5, const uint8\_t\* inp6, const uint8\_t\* inp7, size\_t inlen)

void shake512x8(uint8\_t\* out0, uint8\_t\* out1, uint8\_t\* out2, uint8\_t\* out3,

uint8\_t\* out4, uint8\_t\* out5, uint8\_t\* out6, uint8\_t\* out7, size\_t outlen,

const uint8\_t\* inp0, const uint8\_t\* inp1, const uint8\_t\* inp2, const uint8\_t\* inp3,

const uint8\_t\* inp4, const uint8\_t\* inp5, const uint8\_t\* inp6, const uint8\_t\* inp7, size\_t inlen)

**8.2** **cSHAKE**

**Compute**

The compute function is the short form of the API, the key, custom and name arrays are added and the output is processed using a single function call. The cSHAKE compute functions take as parameters; a pointer to the output array, the number of bytes to write to that array, a pointer to the key array, and the keys byte length, pointers to the name and custom arrays, and their lengths.

void qsc\_cshake128\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* name, size\_t namelen, const uint8\_t\* custom, size\_t custlen)

void qsc\_cshake256\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* name, size\_t namelen, const uint8\_t\* custom, size\_t custlen)

void qsc\_cshake512\_compute(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* name, size\_t namelen, const uint8\_t\* custom, size\_t custlen)

**Initialize**

The initialize function is part of the long-form of the API. It is used to initialize the Keccak state, and inject the key, custom, and name arrays. The cSHAKE initialize function takes a pointer to the Keccak state, the absorption rate, a pointer to the input key array, and the key length as parameters.

void qsc\_cshake\_initialize(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* key, size\_t keylen, const uint8\_t\* name, size\_t namelen, const uint8\_t\* custom, size\_t custlen)

**Squeeze**

The squeeze function outputs pseudo-random in blocks. The block size corresponds to the Keccak rate. Initialize must be called first, and the output array must be sized to receive a rate-sized block of bytes. The squeeze function takes a pointer to the Keccak state, the absorption rate, a pointer to the output byte array, and the number of blocks to write as parameters.

void qsc\_cshake\_squeezeblocks(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, uint8\_t\* output, size\_t nblocks)

**Update**

The update function can inject new keying or message material into the Keccak state. The update function takes the Keccak state, the absorption rate, a pointer to the key array, and the key length as parameters.

void qsc\_cshake\_update(qsc\_keccak\_state\* ctx, qsc\_keccak\_rate rate, const uint8\_t\* key, size\_t keylen)

**References:**

* NIST SHA3 FIPS 202 <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>

**8.3** **HKDF**

**Header:**

sha2.h

**Description:**

Hash-based Key Derivation Function, is a key stretching function, in this implementation it uses either the SHA2-256, or SHA2-512 hashes as the underlying pseudo-random function.

**API:**

**Expand**

The expand function is a key-expansion function. The expand function takes a pointer to the output array, the number of bytes to write to that array, a pointer to key and optional info arrays, and their lengths as parameters.

void qsc\_hkdf256\_expand(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* info, size\_t infolen)

void qsc\_hkdf512\_expand(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* info, size\_t infolen)

**Extract**

The extract function combines multiple inputs and extracts a pseudo-random output. The extract function takes a pointer to the output array, the output length, and pointers to a key and optional salt array, and their lengths as parameters.

void qsc\_hkdf256\_extract(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* salt, size\_t saltlen)

void qsc\_hkdf512\_extract(uint8\_t\* output, size\_t outlen, const uint8\_t\* key, size\_t keylen, const uint8\_t\* salt, size\_t saltlen)

**References:**

* RFC 2104: <http://tools.ietf.org/html/rfc2104>
* RFC 5869: <http://tools.ietf.org/html/rfc5869>

**8.4** **CSG**

**Header:**

csg.h

**Description:**

The cSHAKE deterministic random bytes generator (CSG), is an auto re-keying DRBG, that can re-key with random seed material at specified thresholds, to provide forward secrecy and predictive resistance in a long-running DRBG instance.

**Structures:**

The **qsc\_csg\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| kstate | Keccak state | 3264 | Hash state |
| cache | Uint8 Array | 1088 | Cache buffer |
| bctr | Uint64 | 64 | Buffer count |
| cpos | Uint64 | 64 | Cache position |
| crmd | Uint64 | 64 | Cache remainder |
| rate | Uint64 | 64 | Keccak rate |
| pres | Bool | 8 | Auto rekey |

Table 8.4a CSG state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_CSG256\_SEED\_SIZE | 32 | The CSG-256 seed size. |
| QSC\_CSG512\_SEED\_SIZE | 64 | The CSG-512 seed size. |
| QSC\_CSG\_RESEED\_THRESHHOLD | 1024000 | The auto re-seed threshold size, 1MB default. |

Table 8.4b CSG constants

**Call Hierarchy**

The long form of the CSG function API, first requires a call to the initialize function to add the keying material and optional info array. The generate function populates the output array with pseudo-random. The update function can be used to add entropy to the state from an external source, and the dispose function sets the state to default values.

initialize(*state*, *seed*, *seedlen*, *infolen*, *pres*)

generate (*state*, *output*, *outlen*)

update(*state*, *seed*, *seedlen*)

dispose(*state*)

**API:**

**Dispose**

Resets the CSG state members to defaults. Takes a pointer to the CSG state as a parameter.

void qsc\_csg\_dispose(qsc\_csg\_state\* ctx)

**Initialize**

Initializes the generator with seeding material. The initialize function takes a pointer to a seed array and the seed length, a pointer to an optional info parameter and length, and a predictive-resistance Boolean that enables automatic re-seeding.

void qsc\_csg\_initialize(qsc\_csg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen, const uint8\_t\* info, size\_t infolen, bool predres)

**Generate**

Generates an array of pseudo-random and writes it to the output byte array. Takes a pointer to the CSG state, a pointer to the output array, and the output length as parameters.

void qsc\_csg\_generate(qsc\_csg\_state\* ctx, uint8\_t\* output, size\_t outlen)

**Update**

Updates the generator state with new keying material. The update function takes a pointer to the CSG state, a pointer to a seed array, and the seed length as parameters.

void qsc\_csg\_update(qsc\_csg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen)

**References:**

* NIST SHA3 FIPS 202 <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>
* NIST SP800-185: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>

**8.5** **HCG**

**Header:**

hcg.h

**Description:**

HMAC based pseudo-random bytes generator (HCG), uses SHA2-512 internally as the pseudo random function. HCG is an auto re-keying DRBG, that can re-key with random seed material at specified thresholds, to provide forward secrecy and predictive resistance in a long-running DRBG instance.

**Structures:**

The **qsc\_hcg\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| cache | Uint8 Array | 512 | Cache buffer |
| info | Uint8 Array | 448 | Cache buffer |
| nonce | Uint8 Array | 64 | Nonce array |
| bctr | Uint64 | 64 | Buffer count |
| cpos | Uint64 | 64 | Cache position |
| crmd | Uint64 | 64 | Cache remainder |
| pres | Bool | 8 | Auto rekey |

Table 8.5a HCG structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_HCG\_CACHE\_SIZE | 64 | The HCG cache size. |
| QSC\_HCG\_MAX\_INFO\_SIZE | 0x38 | The maximum info parameter size. |
| QSC\_HCG\_NONCE\_SIZE | 0x08 | The nonce array size. |
| QSC\_HCG\_RESEED\_THRESHHOLD | 1024000 | The auto re-seed threshold size. |
| QSC\_HCG\_SEED\_SIZE | 64 | The seed array size. |

Table 8.5b HCG constants

**Call Hierarchy:**

The long form of the HCG function API, first requires a call to the initialize function to add the keying material and optional info array. The generate function populates the output array with pseudo-random. The update function can be used to add additional externally provided entropy to the state, and the dispose function sets the state to default values.

initialize(*state*, *seed*, *seedlen*, *infolen*, *pres*)

generate (*state*, *output*, *outlen*)

update(*state*, *seed*, *seedlen*)

dispose(*state*)

**API:**

**Dispose**

Resets the HCG state members to defaults. Takes the HCG state as a parameter.

void qsc\_hcg\_dispose(qsc\_hcg\_state\* ctx)

**Initialize**

Initializes the generator with seeding material. The initialize function takes a pointer to a seed array and the seed length, a pointer to an optional info parameter and length, and a predictive-resistance Boolean that enables automatic re-seeding.

void qsc\_hcg\_initialize(qsc\_hcg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen, const uint8\_t\* info, size\_t infolen, bool predres)

**Generate**

Generates an array of pseudo-random and writes it to the output. Takes a pointer to the HCG state, a pointer to the output array, and the output length as parameters.

void qsc\_hcg\_generate(qsc\_hcg\_state\* ctx, uint8\_t\* output, size\_t outlen)

**Update**

Updates the generator state with new keying material. The update function takes a pointer to the HCG state, a pointer to a seed array, and the seed length as parameters.

void qsc\_hcg\_update(qsc\_hcg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen)

**References:**

* RFC 2104, HMAC: <http://tools.ietf.org/html/rfc2104>
* FIPS 198-1: <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>

**8.6 SCB**

**Header:**

scb.h

**Description:**

SCB is a cost based key derivation function (KDF) that uses the Keccak cSHAKE XOF function, to produce pseudo-random bytes from a seeded custom SHAKE generator. If a 32-byte key is used, the implementation uses the cSHAKE-256 implementation for pseudo-random generation, if a 64-byte key is used, the generator uses cSHAKE-512. The CPU cost feature is an iteration count in the cost mechanism, it determines the number of times both the state absorption and memory scattering functions execute. The Memory cost, is the maximum number of mebibytes (MiB) the internal cache uses, during execution of the memory cost mechanism. The L2 cache mechanism can be changed to a doubling of 128 KiB (128, 256, 512, 1024 KiB), doubling the cache sizes above the default 256 KiB, also means that the minimum memory cost must be doubled, L2=512 Kib, minimum memcost is 2, 1024 KiB minimum memcost is 4 etc. The generator can be updated with new seed material, which is absorbed into the Keccak state.

**Structures:**

The **qsc\_scb\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| key | Uint8 Array | 512 | Internal key |
| cpuc | Uint64 | 448 | CPU cost |
| memc | Uint64 | 64 | Memory cost |
| klen | Uint64 | 64 | Key length |
| rate | enum | 64 | Keccak rate |

Table 8.6a SCB state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SCB\_256\_SEED\_SIZE | 32 | The 256-bit key size. |
| QSC\_SCB\_512\_SEED\_SIZE | 64 | The 512-bit key size. |
| QSC\_SCB\_L2CACHE\_DEFAULT\_SIZE | 262144 | The default cache size |
| QSC\_SCB\_MEMORY\_COST\_SIZE | 1048576 | The default memory cost size. |
| QSC\_SCB\_MEMORY\_MAXIMUM | 128 | The maximum memory size. |
| QSC\_SCB\_MEMORY\_MINIMUM | 1 | The minimum memory size. |
| QSC\_SCB\_CPU\_MINIMUM | 1 | The minimum CPU cost. |
| QSC\_SCB\_CPU\_MAXIMUM | 1000 | The maximum CPU cost. |

Table 8.6b SCB constants

**API:**

**Dispose**

Resets the SCB state members to defaults. Takes the SCB state as a parameter.

void qsc\_scb\_dispose(qsc\_hcg\_state\* ctx)

**Initialize**

Initializes the generator with seeding material. The initialize function takes a pointer to a seed array and the seed length, a pointer to an optional info parameter and length, and a predictive-resistance Boolean that enables automatic re-seeding.

void qsc\_scb\_initialize(qsc\_hcg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen, const uint8\_t\* info, size\_t infolen, bool predres)

**Generate**

Generates an array of pseudo-random and writes it to the output. Takes a pointer to the SCB state, a pointer to the output array, and the output length as parameters.

void qsc\_scb\_generate(qsc\_hcg\_state\* ctx, uint8\_t\* output, size\_t outlen)

**Update**

Updates the generator state with new keying material. The update function takes a pointer to the SCB state, a pointer to a seed array, and the seed length as parameters.

void qsc\_scb\_update(qsc\_hcg\_state\* ctx, const uint8\_t\* seed, size\_t seedlen)

**8.7** **SecRand**

**Header:**

secrand.h

**Description:**

Secure number generator, that uses the CSG DRBG as the random generator. Provides access to the range of signed and unsigned intergers, from floats and doubles to 8-bit and 64-bit integers. Integer sizes can be scoped to either a maximum value, or maximum and minimum values.

**Structures:**

The **qsc\_secrand\_state** structure contains the internal function state.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| hstate | qsc\_csg\_state | Variable | The CSG DRBG state |
| cache | Uint8 Array | 0x400 | The cache buffer |
| cpos | Uint64 | 0x40 | The cache position |
| init | Boolean | 0x08 | The initialized flag |

Table 8.7a secrand state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SECRAND\_SEED\_SIZE | 0x20 | The expected seed size used in the initialize function |
| QSC\_SECRAND\_CACHE\_SIZE | 0x400 | The size of the internal cache |

Table 8.7b secrand constants

**API:**

**Initialize**

Initialize the random generator with a seed and optional customization array.

void qsc\_secrand\_initialize(const uint8\_t\* seed, size\_t seedlen, const uint8\_t\* custom, size\_t custlen)

**Generate**

Generate an array of pseudo-random bytes.

bool qsc\_secrand\_generate(uint8\_t\* output, size\_t length)

**Next Char**

Generate a signed 8-bit random integer.

int8\_t qsc\_secrand\_next\_char()

**Next Double**

Generate a random double integer.

int8\_t qsc\_secrand\_next\_double()

**Next Int16**

Generate a signed 16-bit random integer.

int16\_t qsc\_secrand\_next\_int16()

**Next Int16 Max**

Generate a signed 16-bit random integer of a maximum value.

int16\_t qsc\_secrand\_next\_int16\_max(int16\_t maximum)

**Next Int16 Max Min**

Generate a signed 16-bit random integer of a maximum and minimum value.

int16\_t qsc\_secrand\_next\_int16\_maxmin(int16\_t maximum, int16\_t minimum)

**Next Uint16**

Generate an unsigned 16-bit random integer.

uint16\_t qsc\_secrand\_next\_uint16()

**Next Uint16 Max**

Generate an unsigned 16-bit random integer of a maximum value.

uint16\_t qsc\_secrand\_next\_uint16\_max(uint16\_t maximum)

**Next Uint16 Max Min**

Generate an unsigned 16-bit random integer of a maximum and minimum value.

uint16\_t qsc\_secrand\_next\_uint16\_maxmin(uint16\_t maximum, uint16\_t minimum)

**Next Int32**

Generate a signed 32-bit random integer.

int32\_t qsc\_secrand\_next\_int32()

**Next Int32 Max**

Generate a signed 32-bit random integer of a maximum value.

int32\_t qsc\_secrand\_next\_int32\_max(int32\_t maximum)

**Next Int32 Max Min**

Generate a signed 32-bit random integer of a maximum and minimum value.

int32\_t qsc\_secrand\_next\_int32\_maxmin(int32\_t maximum, int32\_t minimum)

**Next Uint32**

Generate an unsigned 32-bit random integer.

uint32\_t qsc\_secrand\_next\_uint32()

**Next Uint32 Max**

Generate an unsigned 32-bit random integer of a maximum value.

uint32\_t qsc\_secrand\_next\_uint32\_max(uint32\_t maximum)

**Next Uint32 Max Min**

Generate an unsigned 32-bit random integer of a maximum and minimum value.

uint32\_t qsc\_secrand\_next\_uint32\_maxmin(uint32\_t maximum, uint32\_t minimum)

**Next Int64**

Generate a signed 64-bit random integer.

int64\_t qsc\_secrand\_next\_int64()

**Next Int64 Max**

Generate a signed 64-bit random integer of a maximum value.

int64\_t qsc\_secrand\_next\_int64\_max(int64\_t maximum)

**Next Int64 Max Min**

Generate a signed 64-bit random integer of a maximum and minimum value.

int64\_t qsc\_secrand\_next\_int64\_maxmin(int64\_t maximum, int64\_t minimum)

**Next Uint64**

Generate an unsigned 64-bit random integer.

uint64\_t qsc\_secrand\_next\_uint64()

**Next Uint64 Max**

Generate an unsigned 64-bit random integer of a maximum value.

uint64\_t qsc\_secrand\_next\_uint64\_max(uint64\_t maximum)

**Next Uint64 Max Min**

Generate an unsigned 64-bit random integer of a maximum and minimum value.

uint64\_t qsc\_secrand\_next\_uint64\_maxmin(uint64\_t maximum, uint64\_t minimum)

## **9:** **Asymmetric Ciphers**

**9.1** **ECDH**

**Header:**

ecdh.h

**Description:**

The Elliptic Curve Diffie Hellman asymmetric cipher. In a Diffie Hellman key exchange, parties generate key-pairs, and exchange their public keys. A shared secret is derived by both parties combing the received public key with their private keys.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_ECDH\_PRIVATEKEY\_SIZE | 32 | The byte size of the secret private-key array. |
| QSC\_ECDH\_PUBLICKEY\_SIZE | 32 | The byte size of the public-key array. |
| QSC\_ECDH\_SHAREDSECRET\_SIZE | 32 | The byte size of the shared secret-key array. |
| QSC\_ECDH\_SEED\_SIZE | 32 | The byte size of the shared secret-key array. |

Table 9.1a ECDH constants

**API:**

**Key Exchange**

Hosts exchange public keys, then combine them with their private keys to extract a shared secret. Takes a pointer to the secret output array, the private key, and public key, as parameters, and returns true if the shared-secret was extracted successfully, or false if the key exchange fails.

bool qsc\_ecdh\_key\_exchange(uint8\_t\* secret, const uint8\_t\* privatekey, const uint8\_t\* publickey)

**Generate**

Generates a public/private key-pair used in the derivation of a shared secret. The generate function takes pointers to the output public key, the private key, and a pointer to the RNG function as parameters.

void qsc\_ecdh\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

void qsc\_ecdh\_generate\_seeded\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, const uint8\_t\* seed)

**References:**

NaCI by Daniel J. Bernstein, Tanja Lange, Peter Schwabe <https://nacl.cr.yp.to>

**9.2** **Kyber**

**Header:**

kyber.h

**Description:**

The NIST Post Quantum Competition round 3 asymmetric cipher finalist Kyber. Kyber encapsulates a shared secret with the public key, which is extracted from the cipher-text using the private key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_KYBER\_CIPHERTEXT\_SIZE | variable | The byte size of the cipher-text array. |
| QSC\_KYBER\_PRIVATEKEY\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_KYBER\_PUBLICKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_KYBER\_SEED\_SIZE | 32 | The byte size of the seed array. |
| QSC\_KYBER\_SHAREDSECRET\_SIZE | 32 | The byte size of the shared secret-key array. |

Table 9.2a Kyber constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_KYBER\_S3Q3329N256K3 | N/A | The Kyber S3Q3329N256K3 parameter set. |
| QSC\_KYBER\_S5Q3329N256K4 | N/A | The Kyber S5Q3329N256K4 parameter set. |
| QSC\_KYBER\_S6Q3329N256K5 | N/A | The Kyber S6Q3329N256K5 parameter set. |

Table 9.2b Kyber parameter sets

**API:**

**Decapsulate**

Decapsulates the shared secret from the cipher-text. The decapsulate function takes pointers to the shared-secret array, and the cipher-text array, and a pointer to the private key as parameters, and returns true if the cipher-text was decapsulated successfully, or false if the key exchange fails.

bool qsc\_kyber\_decapsulate(uint8\_t\* secret, const uint8\_t\* ciphertext, const uint8\_t\* privatekey)

**Encapsulate**

Encapsulates a shared secret in cipher-text, using the public key. The encapsulate function takes a pointer to the shared-secret array, the cipher-text array, and the public key, and a pointer to an RNG function.

void qsc\_kyber\_encapsulate(uint8\_t\* secret, uint8\_t\* ciphertext, const uint8\_t\* publickey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Generate**

The generate function creates a public and private key-pair. It takes pointers to the public and private key arrays, and a pointer to an RNG function.

void qsc\_kyber\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**References:**

* Kyber a CCA-secure module-lattice-based KEM: <https://eprint.iacr.org/2017/634.pdf>
* Kyber, a simple, provably secure key exchange: <http://eprint.iacr.org/2012/688.pdf>

**9.3** **McEliece**

**Description:**

The NIST Post Quantum Competition round 3 asymmetric cipher finalist Classic McEliece. Encapsulates a shared secret with the public key, which is extracted from the cipher-text using the private key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_MCELIECE\_CIPHERTEXT\_SIZE | variable | The byte size of the cipher-text array. |
| QSC\_ MCELIECE \_PRIVATEKEY\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_ MCELIECE \_PUBLICKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_ MCELIECE \_SEED\_SIZE | 32 | The byte size of the seed array. |
| QSC\_ MCELIECE \_SHAREDSECRET\_SIZE | 32 | The byte size of the shared secret-key array. |

Table 9.3a McEliece constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_MCELIECE\_S3N4608T96 | N/A | The McEliece S3-N4608T96 parameter set. |
| QSC\_MCELIECE\_S5N6688T128 | N/A | The McEliece S5-N6688T128 parameter set. |
| QSC\_MCELIECE\_S5N6960T119 | N/A | The McEliece S5-N6960T119 parameter set. |
| QSC\_MCELIECE\_S5N8192T128 | N/A | The McEliece S5-N8192T128 parameter set. |

Table 9.3b McEliece parameter sets

**API:**

**Decapsulate**

Decapsulates the shared secret from the cipher-text. The decapsulate function takes a pointer to the shared-secret array, and the cipher-text array, and a pointer to the private key as parameters, and returns true if the cipher-text was decapsulated successfully, or false if the key exchange fails.

bool qsc\_mceliece\_decapsulate(uint8\_t\* secret, const uint8\_t\* ciphertext, const uint8\_t\* privatekey)

**Encapsulate**

Encapsulates a shared secret in cipher-text using the public key. The encapsulate function takes a pointer to the shared-secret array, the cipher-text array, and the public key, and a pointer to an RNG function as parameters.

void qsc\_mceliece\_encapsulate(uint8\_t\* secret, uint8\_t\* ciphertext, const uint8\_t\* publickey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Generate**

The generate function creates a public and private key-pair. It takes pointers to the public and private key arrays, and a pointer to an RNG function.

void qsc\_mceliece\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**References:**

* Classic McEliece: <https://classic.mceliece.org/nist/mceliece-20171129.pdf>

**9.4** **NTRU**

**Header:**

ntru.h

**Description:**

The NIST Post Quantum Competition round 3 asymmetric cipher finalist, NTRU. Encapsulates a shared secret with the public key, which is extracted from the cipher-text using the private key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_NTRU\_CIPHERTEXT\_SIZE | variable | The byte size of the cipher-text array. |
| QSC\_ NTRU \_PRIVATEKEY\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_ NTRU \_PUBLICKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_ NTRU \_SEED\_SIZE | 32 | The byte size of the seed array. |
| QSC\_ NTRU \_SHAREDSECRET\_SIZE | 32 | The byte size of the shared secret-key array. |

Table 9.4a NTRU constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_NTRU\_S1HPS2048509 | N/A | The NTRU S1HPS2048509 parameter set. |
| QSC\_NTRU\_HPSS32048677 | N/A | The NTRU HPSS32048677 parameter set. |
| QSC\_NTRU\_S5HPS4096821 | N/A | The NTRU S5HPS4096821 parameter set. |
| QSC\_NTRU\_S5HRSS701 | N/A | The NTRU S5HRSS701 parameter set. |

Table 9.4b NTRU parameter sets

**API:**

**Decapsulate**

Decapsulates the shared secret from the cipher-text. The decapsulate function takes a pointer to the shared-secret array, and the cipher-text array, and a pointer to the private key as parameters, and returns true if the cipher-text was decapsulated successfully, or false if the key exchange fails.

bool qsc\_ntru\_decapsulate(uint8\_t\* secret, const uint8\_t\* ciphertext, const uint8\_t\* privatekey)

**Encapsulate**

Encapsulates a shared secret in cipher-text using the public key. The encapsulate function takes a pointer to the shared-secret array, the cipher-text array, and the public key, and a pointer to an RNG function.

void qsc\_ntru\_encapsulate(uint8\_t\* secret, uint8\_t\* ciphertext, const uint8\_t\* publickey, bool (\*rng\_generate)(uint8\_t\*, size\_t));

**Generate**

The generate function creates a public and private key-pair. The generate function takes pointers to the public and private key arrays, and a pointer to an RNG function as parameters.

void qsc\_ntru\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**References:**

The NTRU Algorithm: <https://ntru.org/f/ntru-20190330.pdf>

## **10:** **Asymmetric Signature Schemes**

**10.1** **Dilithium**

**Header:**

dilithium.h

**Description:**

The NIST Post Quantum Competition round 3 asymmetric signature-scheme finalist Dilithium. A message is signed using the private key, and verified using the public key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_DILITHIUM\_PRIVATEKEY\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_DILITHIUM\_PUBLICKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_DILITHIUM\_SIGNATURE\_SIZE | variable | The byte size of the signature array. |

Table 10.1a Dilithium constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_DILITHIUM\_S2N256Q8380417K4 | N/A | The Dilithium S1N256Q8380417 parameter set. |
| QSC\_DILITHIUM\_S3N256Q8380417K6 | N/A | The Dilithium S2N256Q8380417 parameter set. |
| QSC\_DILITHIUM\_S5N256Q8380417K8 | N/A | The Dilithium S3N256Q8380417 parameter set. |

Table 10.1b Dilithium parameter sets

**API:**

**Generate**

The generate function creates the public verification key and private signing key.

The generate function takes pointers to the public and private key arrays, and a pointer to the RNG function as parameters.

void qsc\_dilithium\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Sign**

The sign function signs a message. It takes pointers to the signed message array and signed message length, a pointer to the message array, the message size, a pointer to the private key, and a pointer to the RNG function as parameters.

void qsc\_dilithium\_sign(uint8\_t\* signedmsg, size\_t\* smsglen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Verify**

The verify function, authenticates the signature attached to the message. The verify function takes a pointer to the output message array and message length, a pointer to the signed message, and public key array as parameters, and returns true if the signature is verified, false if it has failed.

bool qsc\_dilithium\_verify(uint8\_t\* message, size\_t\* msglen, const uint8\_t\* signedmsg, size\_t smsglen, const uint8\_t\* publickey)

**References:**

* Dilithium Reference Paper : <https://pq-crystals.org/dilithium/data/dilithium-specification.pdf>
* Dilithium, A Lattice-Based Digital Signature Scheme: <https://pq-crystals.org/dilithium/data/dilithium-20180114.pdf>

**10.2** **ECDSA**

**Header:**

ecdsa.h

**Description:**

The Elliptic Curve Digital Signature Algorithm, signs a message that can be verified with a public key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_ECDSA\_SIGNATURE\_SIZE | 64 | The byte size of the signature array. |
| QSC\_ECDSA\_PRIVATEKEY\_SIZE | 64 | The byte size of the secret private-key array. |
| QSC\_ECDSA\_PUBLICKEY\_SIZE | 32 | The byte size of the public-key array. |
| QSC\_ECDSA\_SEED\_SIZE | 32 | The byte size of the random seed array. |

Table 10.2a ECDSA constants

**API:**

**Generate**

The generate function creates the public verification key and private signing key. The function takes pointers to the public and private key arrays, and a pointer to the RNG function as parameters.

void qsc\_ecdsa\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Sign**

The sign function signs a message. It takes pointers to the signed message array and signed message length, a pointer to the message array, the message size, and a pointer to the private key as parameters.

void qsc\_ecdsa\_sign(uint8\_t\* signedmsg, size\_t\* smsglen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* privatekey)

**Verify**

The verify function, authenticates the signature attached to the message. The verify function takes a pointer to the output message array and message length, a pointer to the signed message, and public key array as parameters, and returns true if the signature is verified, false if it has failed.

bool qsc\_ecdsa\_verify(uint8\_t\* message, size\_t\* msglen, const uint8\_t\* signedmsg, size\_t smsglen, const uint8\_t\* publickey)

**References:**

NaCI by Daniel J. Bernstein, Tanja Lange, Peter Schwabe <https://nacl.cr.yp.to>

**9.3** **Falcon**

**Header:**

falcon.h

**Description:**

The NIST Post Quantum Competition round 3 asymmetric signature-scheme finalist Falcon. A message is signed using the private key, and verified using the public key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_FALCON\_PRIVATEKEY\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_FALCON\_PUBLICKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_FALCON\_SIGNATURE\_SIZE | variable | The byte size of the signature array. |

Table 10.3a Falcon constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_FALCON\_S3SHAKE256F512 | N/A | The Falcon S3SHAKE256F512 parameter set. |
| QSC\_FALCON\_S5SHAKE256F1024 | N/A | The Falcon S5SHAKE256F1024 parameter set. |

Table 10.3b Falcon parameter sets

**API:**

**Generate**

The generate function creates the public verification key and private signing key.

The function takes pointers to the public and private key arrays, and a pointer to the RNG function.

void qsc\_falcon\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Sign**

The sign function signs a message. It takes pointers to the output signed message array and signed message length, a pointer to the message array, the message size, a pointer to the private key, and a pointer to the RNG function as parameters.

void qsc\_falcon\_sign(uint8\_t\* signedmsg, size\_t\* smsglen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Verify**

The verify function, authenticates the signature attached to the message. The verify function takes a pointer to the output message array and message length, pointers to the signed message, and public key array as parameters, and returns true if the signature is verified, false if it has failed.

bool qsc\_falcon\_verify(uint8\_t\* message, size\_t\* msglen, const uint8\_t\* signedmsg, size\_t smsglen, const uint8\_t\* publickey)

**References:**

The Falcon Algorithm Specification: <https://falcon-sign.info/falcon.pdf>

**10.4** **SPHINCS+**

**Header:**

sphincsplus.h

**Description:**

The NIST Post Quantum Competition round 3 asymmetric signature-scheme candidate SPHINCS+. A message is signed using the private key, and verified using the public key.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SPHINCSPLUS\_SIGNATURE\_SIZE | variable | The byte size of the secret private-key array. |
| QSC\_SPHINCSPLUS\_PRIVATEKEY\_SIZE | variable | The byte size of the public-key array. |
| QSC\_SPHINCSPLUS\_PUBLICKEY\_SIZE | variable | The byte size of the signature array. |

Table 10.4a SPHINCS+ constants

**Parameter Sets:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SPHINCSPLUS\_S3S192SHAKERS | N/A | The SphincsPlus S3S192SHAKERS robust small parameter set. |
| QSC\_SPHINCSPLUS\_S3S192SHAKERF | N/A | The SphincsPlus S3S192SHAKERF robust fast parameter set. |
| QSC\_SPHINCSPLUS\_S5S256SHAKERS | N/A | The SphincsPlus S5S256SHAKERS robust small parameter set. |
| QSC\_SPHINCSPLUS\_S5S256SHAKERF | N/A | The SphincsPlus S5S256SHAKERF robust fast parameter set. |

Table 10.4b SPHINCS+ parameter sets

**API:**

**Generate**

The generate function creates the public verification key and private signing key. The function takes pointers to the public and private key arrays, and a pointer to the RNG function as parameters.

void qsc\_sphincsplus\_generate\_keypair(uint8\_t\* publickey, uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Sign**

The sign function signs a message. It takes pointers to the output signed message array and signed message length, a pointer to the message array, the message size, a pointer to the private key, and a pointer to the RNG function as parameters.

void qsc\_sphincsplus\_sign(uint8\_t\* signedmsg, size\_t\* smsglen, const uint8\_t\* message, size\_t msglen, const uint8\_t\* privatekey, bool (\*rng\_generate)(uint8\_t\*, size\_t))

**Verify**

The verify function, authenticates the signature attached to the message. The verify function takes a pointer to the output message array and message length, a pointer to the signed message, and public key array as parameters, and returns true if the signature is verified, false if it has failed.

bool qsc\_sphincsplus\_verify(uint8\_t\* message, size\_t\* msglen, const uint8\_t\* signedmsg, size\_t smsglen, const uint8\_t\* publickey)

**References:**

* The SPHINCS+ Signature Scheme: <https://sphincs.org/data/sphincs+-specification.pdf>

## **11:** **Networking and Sockets**

**11.1** **IP Info**

**Header:**

ipinfo.h

**Description:**

A set of network address functions, used by the TCP/IP sockets implementation.

**Structures:**

The **qsc\_ipinfo\_ipv4\_address** structure contains the IPv4 address information.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| ipv4 | Uint8 Array | 32 | IPv4 Address |

Table 11.1a IPv4 address structure

The **qsc\_ipinfo\_ipv6\_address** structure contains the IPv4 address information.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| Ipv6 | Uint8 Array | 128 | IPv6 Address |

Table 11.1b IPv6 address structure

The **qsc\_ipinfo\_ipv4\_info** structure contains the IPv4 address information.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| address | IPv4 address | 32 | IPv4 Address |
| port | Uint16 | 16 | Port number |
| mask | Uint8 | 8 | Prefix mask |

Table 11.1c IPv4 address info structure

The **qsc\_ipinfo\_ipv6\_info** structure contains the IPv6 address information.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| address | IPv6 address | 128 | IPv4 Address |
| port | Uint16 | 16 | Port number |
| mask | Uint8 | 0x08 | Prefix mask |

Table 11.1d IPv6 address info structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_IPINFO\_IPV4\_BYTELEN | 0x04 | The ipv4 byte array length. |
| QSC\_IPINFO\_IPV4\_MINLEN | 0x08 | The minimum ipv4 string length. |
| QSC\_IPINFO\_IPV4\_STRNLEN | 0x16 | The ipv4 string length. |
| QSC\_IPINFO\_IPV6\_BYTELEN | 16 | The ipv6 byte array length. |
| QSC\_IPINFO\_IPV6\_STRNLEN | 0x41 | The ipv6 string length. |

Table 11.1e address info constants

**API:**

**Address Any**

Returns an address structure set with the ‘any’ address.

qsc\_ipinfo\_ipv4\_address qsc\_ipinfo\_ipv4\_address\_any()

qsc\_ipinfo\_ipv6\_address qsc\_ipinfo\_ipv6\_address\_any()

**Address Clear**

Resets an address structure to zero.

void qsc\_ipinfo\_ipv4\_address\_clear(qsc\_ipinfo\_ipv4\_address\* address)

void qsc\_ipinfo\_ipv6\_address\_clear(qsc\_ipinfo\_ipv6\_address\* address)

**Address From Array**

Instantiate an IP address structure using a serialized 8-bit integer array.

qsc\_ipinfo\_ipv4\_address qsc\_ipinfo\_ipv4\_address\_from\_array(const uint8\_t\* address)

qsc\_ipinfo\_ipv6\_address qsc\_ipinfo\_ipv6\_address\_from\_array(const uint8\_t\* address)

**Address From Bytes**

Instantiate an ipv4 address structure using a set of 8-bit integers.

qsc\_ipinfo\_ipv4\_address qsc\_ipinfo\_ipv4\_address\_from\_bytes(uint8\_t a1, uint8\_t a2, uint8\_t a3, uint8\_t a4)

**Address From String**

Instantiate an IP address structure using a serialized address string.

qsc\_ipinfo\_ipv4\_address qsc\_ipinfo\_ipv4\_address\_from\_string(const char input[QSC\_IPINFO\_IPV4\_STRNLEN])

qsc\_ipinfo\_ipv6\_address qsc\_ipinfo\_ipv6\_address\_from\_string(const char input[QSC\_IPINFO\_IPV6\_STRNLEN])

**Address Is Equal**

Compares two IP address structures for equivalence.

bool qsc\_ipinfo\_ipv4\_address\_is\_equal(const qsc\_ipinfo\_ipv4\_address\* a, const qsc\_ipinfo\_ipv4\_address\* b)

bool qsc\_ipinfo\_ipv6\_address\_is\_equal(const qsc\_ipinfo\_ipv6\_address\* a, const qsc\_ipinfo\_ipv6\_address\* b)

**Address Is Routable**

Tests that the IP address is a valid publicly routable address.

bool qsc\_ipinfo\_ipv4\_address\_is\_routable(const qsc\_ipinfo\_ipv4\_address\* address)

bool qsc\_ipinfo\_ipv6\_address\_is\_routable(const qsc\_ipinfo\_ipv6\_address\* address)

**Address Is Valid**

Tests the IP address for validity and proper format.

bool qsc\_ipinfo\_ipv4\_address\_is\_valid(const qsc\_ipinfo\_ipv4\_address\* address)

bool qsc\_ipinfo\_ipv6\_address\_is\_valid(const qsc\_ipinfo\_ipv6\_address\* address)

**Address Is Zeroed**

Tests that the IP address is in an un-initialized state.

bool qsc\_ipinfo\_ipv4\_address\_is\_zeroed(const qsc\_ipinfo\_ipv4\_address\* address)

bool qsc\_ipinfo\_ipv6\_address\_is\_zeroed(const qsc\_ipinfo\_ipv6\_address\* address)

**Address Loopback**

Returns a copy of the IP loopback address.

qsc\_ipinfo\_ipv4\_address qsc\_ipinfo\_ipv4\_address\_loop\_back()

qsc\_ipinfo\_ipv6\_address qsc\_ipinfo\_ipv6\_address\_loop\_back()

**Address To Array**

Serializes an IP address structure to a byte array.

void qsc\_ipinfo\_ipv4\_address\_to\_array(uint8\_t\* output, const qsc\_ipinfo\_ipv4\_address\* address)

void qsc\_ipinfo\_ipv6\_address\_to\_array(uint8\_t\* output, const qsc\_ipinfo\_ipv6\_address\* address)

**Address From String**

Serializes an IP address structure to a string.

void qsc\_ipinfo\_ipv4\_address\_to\_string(char output[QSC\_IPINFO\_IPV4\_STRNLEN], const qsc\_ipinfo\_ipv4\_address\* address)

void qsc\_ipinfo\_ipv6\_address\_to\_string(char output[QSC\_IPINFO\_IPV6\_STRNLEN], const qsc\_ipinfo\_ipv6\_address\* address)

**IPv6 Address Type**

Returns the IPv6 address routing prefix-type.

qsc\_ipv6\_address\_prefix\_types qsc\_ipinfo\_ipv6\_address\_type(const qsc\_ipinfo\_ipv6\_address\* address)

**11.2** **Queue**

**Header:**

queue.h

**Description:**

A set of queuing functions, used to queue packets in a network stream.

**Structures:**

The **qsc\_queue\_state** structure contains the queue state variables.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| queue | Uint8 2-D Array | Variable | The queue array |
| tags | Uint64 | 1600 | The tag array |
| count | Uint64 | 64 | The queue count |
| depth | Uint64 | 64 | The queue depth |
| position | Uint64 | 64 | The queue position |

Table 11.2a queue state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_QUEUE\_ALIGNMENT | 64 | The memory alignment constant. |
| QSC\_QUEUE\_MAX\_DEPTH | 64 | The maximum queue depth. |

Table 11.2a queue constants

**API:**

**Destroy**

Resets the queue state structure to zero.

void qsc\_queue\_destroy(qsc\_queue\_state\* ctx)

**Flush**

Flush the contents of the queue to an array.

void qsc\_queue\_flush(qsc\_queue\_state\* ctx, uint8\_t\* output)

**Initialize**

Initialize the queue state.

void qsc\_queue\_initialize(qsc\_queue\_state\* ctx, size\_t depth, size\_t width)

**Items**

Returns the number of items in the queue.

size\_t qsc\_queue\_items(const qsc\_queue\_state\* ctx)

**IsFull**

Returns the ‘full’ status from the queue.

bool qsc\_queue\_isfull(const qsc\_queue\_state\* ctx)

**IsEmpty**

Returns the ‘empty’ status from the queue.

bool qsc\_queue\_isempty(const qsc\_queue\_state\* ctx)

**Pop**

Returns the first member of the queue, and erases that item from the queue.

uint64\_t qsc\_queue\_pop(qsc\_queue\_state\* ctx, uint8\_t\* output, size\_t outlen)

**Push**

Add an item to the queue.

void qsc\_queue\_push(qsc\_queue\_state\* ctx, const uint8\_t\* input, size\_t inplen, uint64\_t tag)

**11.3** **Socket**

**Headers:**

socket.h, socketbase.h, socketflags.h

**Description:**

The TCP/IP network sockets functions.

**Structures:**

The **qsc\_socket** structure contains the IP socket information.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| connection | Uint8 | 0x08 | The packet flag |
| address | Int8 | 0x08 | The address string |
| instance | Uint32 | 0x20 | The instance number |
| port | Uint64 | 0x40 | The port number |
| position | Uint64 | 0x40 | The queue position |
| address\_family | enum | 0x08 | The address family |
| connection\_ status | enum | 0x08 | The connection state |
| socket\_protocol | enum | 0x08 | The socket protocol |
| socket\_transport | enum | 0x08 | The socket transport |

Table 11.3a socket state structure

The **qsc\_socket\_receive\_async** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| callback | Pointer | 0x40 | The return callback |
| error | Pointer | 0x40 | The error callback |
| source | Socket | 0x40 | The socket pointer |
| buffer | Uint8 Array | 0x800 | The message buffer |

Table 11.3b socket async receive structure

The **qsc\_receive\_poll\_state** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| sockarr | Pointer Array | 0x40 | The socket array |
| callback | Pointer | 0x40 | The return callback |
| error | Pointer | 0x40 | The error callback |
| count | Uint64 | 0x40 | The active sockets |

Table 11.3c socket receive poll structure

The **qsc\_socket\_state** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| connection | Socket | 0x40 | The socket pointer |
| address | Int8 | 0x41 | The address string |
| instance | Uint32 | 0x20 | The instance number |
| port | Uint64 | 0x10 | The port number |
| position | Uint64 | 0x40 | The queue position |
| address\_family | enum | 0x08 | The address family |
| connection\_ status | enum | 0x08 | The connection state |
| socket\_protocol | enum | 0x08 | The socket protocol |
| socket\_transport | enum | 0x08 | The socket transport |

Table 11.3d socket state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_UNINITIALIZED\_SOCKET | -1 | An unitialized socket handle. |
| QSC\_SOCKET\_ADDRESS\_MAX\_LENGTH | 0x41 | The address string length. |
| QSC\_SOCKET\_MAX\_CONN | 0x7FFFFFFF | The maximum connections. |
| QSC\_SOCKET\_RET\_ERROR | 0xFFFFFFFF | The base socket error flag. |
| QSC\_SOCKET\_RET\_SUCCESS | 0x00 | The base socket success flag. |
| QSC\_SOCKET\_TERMINATOR\_SIZE | 0x01 | The terminator size. |
| QSC\_SOCKET\_TIMEOUT\_MSEC | 0x6250 | The default wait seconds. |
| QSC\_UNINITIALIZED\_SOCKET | -1 | An unitialized socket handle. |

Table 11.3e socket state constants

The **qsc\_ipv6\_address\_prefix\_types** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_ipv6\_prefix\_none | No prefix is set. |
| qsc\_ipv6\_prefix\_link\_local | An link local address type, not globally routable. |
| qsc\_ipv6\_prefix\_multicast | A qsc\_ipv6\_prefix\_multicast address type, prefix. |
| qsc\_ipv6\_prefix\_global | A globally routable address type, prefix. |
| qsc\_ipv6\_prefix\_unique\_local | A unique local address type, not globally routable, prefix. |

Table 11.3f IPv6 prefix types enumeration

The **qsc\_socket\_address\_families** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_address\_family\_none | No address family is specified AF\_UNSPEC. |
| qsc\_socket\_address\_family\_unix | Unix local to host (pipes, portals) AF\_UNIX. |
| qsc\_socket\_address\_family\_ipv4 | The Internet Protocol 4 address family AF\_INET. |
| qsc\_socket\_address\_family\_ipv6 | The Internet Protocol 6 address family AF\_INET6. |

Table 11.3g address families enumeration

The **qsc\_socket\_states** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_address\_family\_none | No address family is specified AF\_UNSPEC. |
| qsc\_socket\_address\_family\_unix | Unix local to host (pipes, portals) AF\_UNIX. |
| qsc\_socket\_address\_family\_ipv4 | The Internet Protocol 4 address family AF\_INET. |
| qsc\_socket\_address\_family\_ipv6 | The Internet Protocol 6 address family AF\_INET6. |

Table 11.3h socket states enumeration

The **qsc\_socket\_options** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_option\_none | No flag is used. |
| qsc\_socket\_option\_broadcast | Configures a socket for sending broadcast data SO\_BROADCAST. |
| qsc\_socket\_option\_ipv6\_only | Flag used to enable a dual stack configuration IPV6\_V6ONLY. |
| qsc\_socket\_option\_keepalive | Enables sending keep-alive packets for a socket connection SO\_KEEPALIVE. |
| qsc\_socket\_option\_linger | Lingers on close if unsent data is present SO\_LINGER. |
| qsc\_socket\_option\_no\_route | Sets whether outgoing data should be sent on interface the socket is bound to and not a routed on some other interface SO\_DONTROUTE. |
| qsc\_socket\_option\_out\_of\_band | Indicates that out-of-bound data should be returned in-line with regular data SO\_OOBINLINE. |
| qsc\_socket\_option\_reuse\_address | Enables or disables the reuse of a bound address. |
| qsc\_socket\_option\_receive\_time\_out | The timeout, in milliseconds, for blocking received calls SO\_RCVTIMEO. |
| qsc\_socket\_option\_send\_time\_out | The timeout, in milliseconds, for blocking send calls SO\_SNDTIMEO. |
| qsc\_socket\_option\_tcp\_no\_delay | Enables or disables the Nagle algorithm for TCP sockets. This option is disabled (set to FALSE) by default TCP\_NODELAY. |

Table 11.3i socket options enumeration

The **qsc\_socket\_protocols** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_protocol\_none | No protocol type specified. |
| qsc\_socket\_protocol\_ipv4 | Internet Protocol version 4 IPPROTO\_IPV4. |
| qsc\_socket\_protocol\_socket | Enables or disables a socket level option. |
| qsc\_socket\_protocol\_tcp | Transport Control Protocol IPPROTO\_TCP. |
| qsc\_socket\_protocol\_udp | Unreliable Delivery Protocol IPPROTO\_UDP. |
| qsc\_socket\_protocol\_ipv6 | IPv6 header IPPROTO\_IPV6. |
| qsc\_socket\_protocol\_ipv6\_routing | IPv6 Routing header IPPROTO\_ROUTING. |
| qsc\_socket\_protocol\_ipv6\_fragment | IPv6 fragmentation header IPPROTO\_FRAGMENT. |
| qsc\_socket\_protocol\_icmpv6 | ICMPv6 IPPROTO\_ICMPV6. |
| qsc\_socket\_protocol\_ipv6\_no\_header | IPv6 no next header IPPROTO\_NONE. |
| qsc\_socket\_protocol\_dstopts | IPv6 Destination options IPPROTO\_DSTOPTS. |
| qsc\_socket\_protocol\_raw | Raw Packet IPPROTO\_RAW. |

Table 11.3j socket protocols enumeration

The **qsc\_socket\_receive\_flags** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_receive\_flag\_none | No flag is used. |
| qsc\_socket\_receive\_flag\_out\_of\_band | Process out of band data MSG\_OOB. |
| qsc\_socket\_receive\_flag\_peek | Peeks at the incoming data MSG\_PEEK. |
| qsc\_socket\_receive\_flag\_wait\_all | Request completes only when buffer is full MSG\_WAITALL. |

Table 11.3k socket receive enumeration

The **qsc\_socket\_send\_flags** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_send\_flag\_none | No flag is used. |
| qsc\_socket\_send\_flag\_send\_oob | Sends OOB data on a stream type socket MSG\_OOB. |
| qsc\_socket\_send\_flag\_peek\_message | Sends a partial message. |
| qsc\_socket\_send\_flag\_no\_routing | The data packets should not be routed MSG\_DONTROUTE. |

Table 11.3l socket send enumeration

The **qsc\_socket\_shut\_down\_flags** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_shut\_down\_flag\_receive | Shut down the receiving channel QSC\_SOCKET\_SD\_RECEIVE. |
| qsc\_socket\_shut\_down\_flag\_send | Shut down the sending channel QSC\_SOCKET\_SD\_SEND. |
| qsc\_socket\_shut\_down\_flag\_both | Shut down both channels QSC\_SOCKET\_SD\_BOTH. |

Table 11.3m socket shut down enumeration

The **qsc\_socket\_transports** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_socket\_transport\_none | No flag is used. |
| qsc\_socket\_transport\_stream | Streaming connection SOCK\_STREAM. |
| qsc\_socket\_transport\_datagram | Datagram connection SOCK\_DGRAM. |
| qsc\_socket\_transport\_raw | TCP Raw socket SOCK\_RAW. |
| qsc\_socket\_transport\_reliable | Reliable protocol SOCK\_RDM. |
| qsc\_socket\_transport\_sequenced | Sequenced packets SOCK\_SEQPACKET. |

Table 11.3n socket transports enumeration

**API:**

**Exception Callback**

The socket exception callback prototype.

void qsc\_socket\_exception\_callback(qsc\_socket\* source, qsc\_socket\_exceptions error)

**Receive Async Callback**

The socket asynchronous receive callback prototype.

void qsc\_socket\_receive\_async\_callback(qsc\_socket\* source, uint8\_t\* message, size\_t\* msglen)

**Receive Poll Callback**

The receive polling callback prototype.

void qsc\_socket\_receive\_poll\_callback(qsc\_socket\* source, size\_t error)

**IPv4 Valid Address**

Detects if the string contains a valid IPv4 address.

bool qsc\_socket\_ipv4\_valid\_address(const char\* address)

**IPv6 Valid Address**

Detects if the string contains a valid IPv6 address.

bool qsc\_socket\_ipv6\_valid\_address(const char\* address)

**Is Blocking**

Determines if the socket is in blocking mode.

bool qsc\_socket\_is\_blocking(const qsc\_socket\* sock)

**Is Connected**

Determines if the socket is connected.

bool qsc\_socket\_is\_connected(const qsc\_socket\* sock)

**Accept**

Accept function handles an incoming connection attempt on the socket.

qsc\_socket\_exceptions qsc\_socket\_accept(const qsc\_socket\* source, qsc\_socket\* target)

**Attach**

Copy a socket to the target socket.

void qsc\_socket\_attach(qsc\_socket\* source, qsc\_socket\* target)

**Bind**

The Bind function associates an IP address with a socket.

qsc\_socket\_exceptions qsc\_socket\_bind(qsc\_socket\* sock, const char\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_bind\_ipv4(qsc\_socket\* sock, const qsc\_ipinfo\_ipv4\_address\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_bind\_ipv6(qsc\_socket\* sock, const qsc\_ipinfo\_ipv6\_address\* address, uint16\_t port)

**Close**

The Close function closes and disposes of the socket.

qsc\_socket\_exceptions qsc\_socket\_close\_socket(const qsc\_socket\* sock)

**Connect**

The Connect function establishes a connection to a remote host.

qsc\_socket\_exceptions qsc\_socket\_connect(qsc\_socket\* sock, const char\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_connect\_ipv4(qsc\_socket\* sock, const qsc\_ipinfo\_ipv4\_address\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_connect\_ipv6(qsc\_socket\* sock, const qsc\_ipinfo\_ipv6\_address\* address, uint16\_t port)

**Create**

The Create function creates a socket that is bound to a specific transport provider.

qsc\_socket\_exceptions qsc\_socket\_create(qsc\_socket\* sock, qsc\_socket\_address\_families family, qsc\_socket\_transports transport, qsc\_socket\_protocols protocol)

**Listen**

Places the socket in the listening state, waiting for a connection.

qsc\_socket\_exceptions qsc\_socket\_listen(const qsc\_socket\* sock, int32\_t backlog)

**Receive**

Receive data from a synchronous connected socket or a bound connectionless socket.

size\_t qsc\_socket\_receive(const qsc\_socket\* sock, uint8\_t\* output, size\_t outlen, qsc\_socket\_receive\_flags flag)

**Receive Async**

Receive data from a connected socket asynchronously.

qsc\_socket\_exceptions qsc\_socket\_receive\_async(qsc\_socket\_receive\_async\_state\* state)

**Receive All**

Receive a block of data from a synchronous connected socket or a bound connectionless socket, and returns when the buffer is full.

size\_t qsc\_socket\_receive\_all(const qsc\_socket\* sock, uint8\_t\* output, size\_t outlen, qsc\_socket\_receive\_flags flag)

**Receive From**

Receive data from a synchronous connected socket or a bound connectionless socket.

size\_t qsc\_socket\_receive\_from(qsc\_socket\* sock, char\* destination, uint16\_t port, uint8\_t\* output, size\_t outlen, qsc\_socket\_receive\_flags flag)

**Receive Poll**

Polls an array of sockets, and fires a callback if a socket is ready to receive data, or an error if socket is disconnected.

uint32\_t qsc\_socket\_receive\_poll(const qsc\_socket\_receive\_poll\_state\* state)

**Send**

Sends data on a TCP connected socket.

size\_t qsc\_socket\_send(const qsc\_socket\* sock, const uint8\_t\* input, size\_t inlen, qsc\_socket\_send\_flags flag)

**Send To**

Sends data on a UDP socket.

size\_t qsc\_socket\_send\_to(const qsc\_socket\* sock, const char\* destination, size\_t destlen, uint16\_t port, const uint8\_t\* input, size\_t inlen, qsc\_socket\_send\_flags flag)

**Send All**

Sends a block of data larger than a single packet size, on a TCP socket and returns when sent.

size\_t qsc\_socket\_send\_all(const qsc\_socket\* sock, const uint8\_t\* input, size\_t inlen, qsc\_socket\_send\_flags flag)

**Shut Down**

Shuts down a socket.

qsc\_socket\_exceptions qsc\_socket\_shut\_down(qsc\_socket\* sock, qsc\_socket\_shut\_down\_flags parameters)

**Error To String**

Returns the error string associated with the exception code.

const char\* qsc\_socket\_error\_to\_string(qsc\_socket\_exceptions code)

**Last Error**

The last error generated by the internal socket library.

qsc\_socket\_exceptions qsc\_socket\_get\_last\_error()

**IOCTL**

Sets the IO mode of the socket.

qsc\_socket\_exceptions qsc\_socket\_ioctl(const qsc\_socket\* sock, int32\_t command, uint32\_t\* arguments)

**Receive Ready**

Tests the socket to see if it is ready to receive data.

bool qsc\_socket\_receive\_ready(const qsc\_socket\* sock, const struct timeval\* timeout)

**Send Ready**

Tests the socket to see if it is ready to send data.

bool qsc\_socket\_send\_ready(const qsc\_socket\* sock, const struct timeval\* timeout)

**Set Last Error**

Set the last error generated by the socket library.

void qsc\_socket\_set\_last\_error(qsc\_socket\_exceptions error)

**Shut Down Sockets**

Shut down the sockets library.

qsc\_socket\_exceptions qsc\_socket\_shut\_down\_sockets()

**Socket Set Option**

Send an option command to the socket.

qsc\_socket\_exceptions qsc\_socket\_set\_option(const qsc\_socket\* sock, qsc\_socket\_protocols level, qsc\_socket\_options option, int32\_t optval)

**Start Sockets**

Start the sockets library.

bool qsc\_socket\_start\_sockets()

**11.4** **Socket Client**

**Headers:**

socketclient.h, socket.h, socketbase.h, socketflags.h

**Description:**

The socket client functions.

**API:**

**Address Family**

Returns the sockets address family, IPv4 or IPv6.

qsc\_socket\_address\_families qsc\_socket\_client\_address\_family(const qsc\_socket\* sock)

**Socket Protocol**

Returns the socket protocol type.

qsc\_socket\_protocols qsc\_socket\_client\_socket\_protocol(const qsc\_socket\* sock)

**Connect Host**

Connect to a remote host using the network host name and service name.

qsc\_socket\_exceptions qsc\_socket\_client\_connect\_host(qsc\_socket\* sock, const char\* host, const char\* service)

qsc\_socket\_exceptions qsc\_socket\_client\_connect\_ipv4(qsc\_socket\* sock, const qsc\_ipinfo\_ipv4\_address\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_client\_connect\_ipv6(qsc\_socket\* sock, const qsc\_ipinfo\_ipv6\_address\* address, uint16\_t port)

**Socket Transport**

Get the socket transport type.

qsc\_socket\_transports qsc\_socket\_client\_socket\_transport(const qsc\_socket\* sock)

**Client Initialize**

Initialize the server socket.

void qsc\_socket\_client\_initialize(qsc\_socket\* sock)

**Client Receive**

Receive data from a synchronous connected socket or a bound connectionless socket.

size\_t qsc\_socket\_client\_receive(const qsc\_socket\* sock, char\* output, size\_t outlen, qsc\_socket\_receive\_flags flag)

**Client Receive From**

Receive UDP data from a remote host.

size\_t qsc\_socket\_client\_receive\_from(qsc\_socket\* sock, char\* address, uint16\_t port, char\* output, size\_t outlen, qsc\_socket\_receive\_flags flag)

**Client Send**

Sends data on a connected socket.

size\_t qsc\_socket\_client\_send(const qsc\_socket\* sock, const char\* input, size\_t inplen, qsc\_socket\_send\_flags flag)

**Client Send To**

Sends UDP data to a remote host.

size\_t qsc\_socket\_client\_send\_to(qsc\_socket\* sock, const char\* address, uint16\_t port, const char\* input, size\_t inplen, qsc\_socket\_send\_flags flag)

**Client Shut Down**

Shut down the socket.

void qsc\_socket\_client\_shut\_down(qsc\_socket\* sock)

**11.5** **Socket Server**

**Headers:**

socketserver.h, socket.h, socketbase.h, socketflags.h

**Description:**

The socket server function definitions.

**Structures:**

The **qsc\_socket\_server\_accept\_result** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| target | Socket | 64 | The socket pointer |

Table 11.5a socket async receive structure

The **qsc\_socket\_server\_async\_accept\_state** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| callback | Pointer | 64 | The return callback |
| error | Pointer | 64 | The error callback |
| source | Socket | 64 | The socket pointer |

Table 11.5b socket async accept structure

**API:**

**Accept Callback**

The socket server accept callback prototype.

void qsc\_socket\_server\_accept\_callback(qsc\_socket\_server\_accept\_result\* ares)

**Error Callback**

The socket server error callback prototype.

void qsc\_socket\_server\_error\_callback(qsc\_socket\* source, qsc\_socket\_exceptions error)

**Address Family**

Returns the sockets address family, IPv4 or IPv6.

qsc\_socket\_address\_families qsc\_socket\_server\_address\_family(const qsc\_socket\* sock)

**Socket Protocol**

Returns the socket protocol type.

qsc\_socket\_protocols qsc\_socket\_server\_socket\_protocol(const qsc\_socket\* sock)

**Socket Transport**

Returns the socket transport type.

qsc\_socket\_transports qsc\_socket\_server\_socket\_transport(const qsc\_socket\* sock)

**Close Socket**

Close the socket.

void qsc\_socket\_server\_close\_socket(qsc\_socket\* sock)

**Initialize**

Initialize the server socket.

void qsc\_socket\_server\_initialize(qsc\_socket\* sock)

**Listen**

Places the source socket in a blocking listening state, and waits for a connection.

qsc\_socket\_exceptions qsc\_socket\_server\_listen(qsc\_socket\* source, qsc\_socket\* target, const char\* address, uint16\_t port, qsc\_socket\_address\_families family)

qsc\_socket\_exceptions qsc\_socket\_server\_listen\_ipv4(qsc\_socket\* source, qsc\_socket\* target, const qsc\_ipinfo\_ipv4\_address\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_server\_listen\_ipv6(qsc\_socket\* source, qsc\_socket\* target, const qsc\_ipinfo\_ipv6\_address\* address, uint16\_t port)

**Listen Async**

Places the socket in an asynchronous listening state.

qsc\_socket\_exceptions qsc\_socket\_server\_listen\_async(qsc\_socket\_server\_async\_accept\_state\* state, const char\* address, uint16\_t port, qsc\_socket\_address\_families family)

qsc\_socket\_exceptions qsc\_socket\_server\_listen\_async\_ipv4(qsc\_socket\_server\_async\_accept\_state\* state, const qsc\_ipinfo\_ipv4\_address\* address, uint16\_t port)

qsc\_socket\_exceptions qsc\_socket\_server\_listen\_async\_ipv6(qsc\_socket\_server\_async\_accept\_state\* state, qsc\_ipinfo\_ipv6\_address\* address, uint16\_t port)

**Set Option**

Send an option command to the socket.

void qsc\_socket\_server\_set\_options(const qsc\_socket\* sock, qsc\_socket\_protocols level, qsc\_socket\_options option, int32\_t optval)

**Shut Down**

Shut down the server.

void qsc\_socket\_server\_shut\_down(qsc\_socket\* sock)

**11.6** **Network Utilities**

**Header:**

netutils.h

**Description:**

The network utilities functions, are used to access network services and information.

**Structures:**

The **qsc\_netutils\_adaptor\_info** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| desc | char | 0x420 | The description string |
| dhcp | char | 0x41 | The dhcp string |
| gateway | char | 32 | The gateway string |
| ip | char | 16 | The port number |
| mac | char | 64 | The queue position |
| name | char | 0x08 | The address family |
| subnet | char | 0x08 | The connection state |

Table 11.6a network adaptor info structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_NET\_MAC\_ADAPTOR\_NAME | 0x104 | The network adaptors info string. |
| QSC\_NET\_MAC\_ADAPTOR\_DESCRIPTION | 0x84 | The network adaptors description string. |
| QSC\_NET\_MAC\_ADAPTOR\_INFO\_ARRAY | 0x08 | The network adaptors info array size. |
| QSC\_NET\_IP\_STRING\_SIZE | 0x80 | The ip address string size. |
| QSC\_NET\_HOSTS\_NAME\_BUFFER | 0x104 | The size of the hosts name buffer. |
| QSC\_NET\_MAC\_ADDRESS\_LENGTH | 0x08 | The mac address buffer length. |
| QSC\_NET\_PROTOCOL\_NAME\_BUFFER | 0x80 | The size of the protocol name buffer. |
| QSC\_NET\_SERVICE\_NAME\_BUFFER | 0x80 | The size of the service name buffer. |

Table 11.6b network utilities constants

**API:**

**Get Adaptor Info**

Retrieves the address information and the first addressable interface. Takes the netutils\_adaptor\_info structure as the parameter.

void qsc\_netutils\_get\_adaptor\_info(qsc\_netutils\_adaptor\_info\* info)

**Get Adaptor Info Array**

Retrieves the address information and all addressable interfaces. Takes the netutils\_adaptor\_info structure array as the parameter.

void qsc\_netutils\_get\_adaptor\_info\_array(qsc\_netutils\_adaptor\_info ctx[QSC\_NET\_MAC\_ADAPTOR\_INFO\_ARRAY])

**Get Domain Name**

Retrieves the hosts domain name. Takes a pointer to a char array as the parameter, and returns the domain name size.

size\_t qsc\_netutils\_get\_domain\_name(char output[QSC\_NET\_HOSTS\_NAME\_BUFFER])

**Get IPv4 Address**

Gets the first IPv4 addressable interface address. Returns the IPv4 address structure.

qsc\_ipinfo\_ipv4\_address qsc\_netutils\_get\_ipv4\_address()

**Get IPv6 Address**

Gets the first IPv6 addressable interface address. Returns the IPv6 address structure.

qsc\_ipinfo\_ipv6\_address qsc\_netutils\_get\_ipv6\_address()

**Get IPv4 Info**

Retrieves the IPv4 address information for a remote host. Takes the host name array, and the service name string as parameters, and returns an IPv4 address info structure.

qsc\_ipinfo\_ipv4\_info qsc\_netutils\_get\_ipv4\_info(const char host[QSC\_NET\_HOSTS\_NAME\_BUFFER], const char service[QSC\_NET\_SERVICE\_NAME\_BUFFER])

**Get IPv6 Info**

Retrieves the IPv6 address information for a remote host. Takes the host name array, and the service name string as parameters, and returns an IPv6 address info structure.

qsc\_ipinfo\_ipv6\_info qsc\_netutils\_get\_ipv6\_info(const char host[QSC\_NET\_HOSTS\_NAME\_BUFFER], const char service[QSC\_NET\_SERVICE\_NAME\_BUFFER])

**Get MAC Address**

Retrieves the MAC address of the first addressable interface. Takes the MAC address array as the parameter.

## void qsc\_netutils\_get\_mac\_address(uint8\_t mac[QSC\_NET\_MAC\_ADDRESS\_LENGTH])

**Get Peer Name**

Retrieves the host name of the connected peer. Takes the output char array, and a pointer to the socket structure as parameters.

void qsc\_netutils\_get\_peer\_name(char output[QSC\_NET\_HOSTS\_NAME\_BUFFER], const qsc\_socket\* sock)

**Get Socket Name**

Retrieves the socket name of the connected peer. Takes the output array and a pointer to the connected socket structure as parameters.

void qsc\_netutils\_get\_socket\_name(char output[QSC\_NET\_PROTOCOL\_NAME\_BUFFER], const qsc\_socket\* sock)

**Port Name To Number**

Get the port number from the service name. Takes the port name, and the protocol as parameters, and returns the port number.

uint16\_t qsc\_netutils\_port\_name\_to\_number(const char portname[QSC\_NET\_HOSTS\_NAME\_BUFFER], const char protocol[QSC\_NET\_PROTOCOL\_NAME\_BUFFER])

**12:** **Threads and Asynchronous Processing**

**12.1** **Asynchronous Threading**

**Header:**

async.h

**Description:**

Asynchronous threading and mutex functions.

**Types:**

|  |  |
| --- | --- |
| Type Name | Purpose |
| qsc\_async\_mutex | The thread mutex type. |
| qsc\_thread | The thread instance type. |

Table 12.1a async type definitions

**API:**

**Launch Thread**

Launch a function on a new thread. Takes a pointer to the function, and a state containing the function parameters.

void qsc\_async\_launch\_thread(void (\*func)( void\*), void\* state)

**Launch Parallel Threads**

Launch a series of threads, using variadic function arguments. Takes a pointer to the function, the parameter count, and a variadic argument for parameters.

void qsc\_async\_launch\_parallel\_threads(void (\*func) (void\*), size\_t count, ...)

**Mutex Create**

Create a new mutex instance. Takes a pointer to the mutex as a parameter, and returns true if the mutex was acquired successfully.

bool qsc\_async\_mutex\_create(qsc\_async\_mutex\* mtx)

**Mute Destroy**

Destroy a mutex instance, and set the value to zero. Takes a pointer to the mutex as a parameter, and returns true if the mutex was destroyed successfully.

bool qsc\_async\_mutex\_destroy(qsc\_async\_mutex\* mtx)

**Mutex Lock**

Lock a mutex, creating the thread barrier. Takes a pointer to the mutex as a parameter.

void qsc\_async\_mutex\_lock(qsc\_async\_mutex\* mtx)

**Mutex Lock Ex**

Initializes and locks a mutex in a single operation. Takes a pointer to the mutex as a parameter.

void qsc\_async\_mutex\_lock\_ex(qsc\_async\_mutex\* mtx)

**Mutex Unlock**

Unlock a mutex, releasing the thread barrier. Takes a pointer to the mutex as a parameter.

void qsc\_async\_mutex\_unlock(qsc\_async\_mutex\* mtx)

**Mutex Unlock Ex**

Unlocks a mutex, releasing the thread barrier, and destroys the mutex with a single operation. Takes a pointer to the mutex as a parameter.

void qsc\_async\_mutex\_unlock\_ex(qsc\_async\_mutex\* mtx)

**Parallel For**

A parallel for loop implementation. Processes each function call on a unique thread and waits for all threads before returning.

bool qsc\_async\_parallel\_for(void (\*task)(void \*context, size\_t index), void \*context, size\_t nthreads)

**Processor Count**

Get the number of virtual processors on the system.

size\_t qsc\_async\_processor\_count()

**Thread Create**

Create a thread with a single parameter. Takes a function pointer, and a pointer to the function state structure as parameters, and returns the thread handle, or zero on failure.

qsc\_thread qsc\_async\_thread\_create(void (\*func)(void\*), void\* state)

**Thread CreateEx**

Create a thread with multiple parameters using a variadic expression. Takes a pointer to the function, the parameter count, and a variadic argument for parameters, and returns the thread handle, or zero on failure.

qsc\_thread qsc\_async\_thread\_create\_ex(void (\*func)(void\*\*), void\*\* args)

**Thread Resume**

Resume a suspended thread. Takes the thread as a parameter and returns zero on success.

int32\_t qsc\_async\_thread\_resume(qsc\_thread\* handle)

**Thread Sleep**

Pause the thread for a number of milliseconds. Takes the number of milliseconds as a parameter.

void qsc\_async\_thread\_sleep(uint32\_t msec)

**Thread Suspend**

Suspend a thread. Takes the thread as a parameter and returns zero on success.

int32\_t qsc\_async\_thread\_suspend(qsc\_thread\* handle)

**Thread Terminate**

Terminate a thread, and release the memory. Takes the thread handle as a parameter.

void qsc\_async\_thread\_terminate(qsc\_thread\* handle)

**Thread Wait**

Wait for a thread to complete execution. Takes the thread handle as a parameter.

void qsc\_async\_thread\_wait(qsc\_thread\* handle)

**Thread Wait Time**

Wait for a thread for a maximum length of time. Takes the thread handle, and the number of millisecons as parameters.

void qsc\_async\_thread\_wait\_all(qsc\_thread\* handle, uint32\_t msec)

**Thread Wait All**

Wait for an array of threads to complete execution. Takes the pointer to an array of thread handles, and the number of handles as parameters.

void qsc\_async\_thread\_wait\_all(qsc\_thread\* handles, int32\_t count)

**12.2** **Events**

**Header:**

event.h

**Description:**

A set of functions used for event-style callback interfaces.

**Types:**

|  |  |
| --- | --- |
| Type Name | Purpose |
| qsc\_event\_callback | The event callback function prototype. |

Table 12.2a event types

**Enumerations:**

The **qsc\_event\_list** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_event\_receive\_callback | A receive function callback. |
| qsc\_event\_send\_callback | A send function callback. |
| qsc\_event\_connection\_request | A connection request callback. |
| qsc\_event\_connection\_shutdown | A shutdown request callback. |

Table 12.2b event list enumeration

**Structures:**

The **qsc\_event\_handlers** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| callback | qsc\_event\_callback | 0x40 | The callback function |
| next | qsc\_event\_handlers | 0x40 | The dhcp string |

Table 12.2c event handler structure

**API:**

**Event Register**

Register an event and callback. Takes the event enumerator, and the event callback as parameters, returns zero for success.

int32\_t qsc\_event\_register(qsc\_event\_list event, qsc\_event\_callback callback)

**Initialize Listeners**

Initialize the event handler array. Takes an array of event handler structures as a parameter.

void qsc\_event\_init\_listeners(qsc\_event\_handlers\* handlers[QSC\_EVENT\_LIST\_LENGTH])

**Get Callback**

Retreive an event callback by name. Takes the event callback name and returns the callcack function handle.

qsc\_event\_callback qsc\_event\_get\_callback(const char name[QSC\_EVENT\_NAME])

**Destroy Listeners**

Destroy the event handler array. Takes an array of event handler structures as a parameter.

void qsc\_event\_destroy\_listeners(qsc\_event\_handlers\* handlers[QSC\_EVENT\_LIST\_LENGTH]

**12.3 Thread Pool**

**Header:**

threadpool.h

**Description:**

Thread-pool creation and management functions.

**Structures:**

The **qsc\_threadpool\_state** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| tpool | pthread\_t Array | 0x512 | The thread array |
| tcount | Uint64 | 0x40 | The thread count |

Table 12.4a thread pool structure

**API:**

**Add Task**

Add a task to the threadpool. Takes a pointer to the threadpool, and a function pointer as parameters, and returns true on success.

bool qsc\_threadpool\_add\_task(qsc\_threadpool\_state\* ctx, void (\*thd\_func)(void\*), void\* state)

**Clear**

Clear all threads from the threadpool. Takes a pointer to threadpool as a parameter.

void qsc\_threadpool\_clear(qsc\_threadpool\_state\* ctx)

**Initialize**

Initializes the threadpool structure. Takes a pointer to the threadpool as a parameter.

void qsc\_threadpool\_initialize(qsc\_threadpool\_state\* ctx)

**Remove Task**

Remove a task from the threadpool. Takes a pointer to the threadpool and the thread index as parameters.

void qsc\_threadpool\_remove\_task(qsc\_threadpool\_state\* ctx, size\_t index)

**13:** **Storage and Data Processing**

**13.1** **Collection**

**Header:**

collection.h

**Description:**

A keyed collection implementation.

**Structures:**

The **qsc\_collection\_state** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| items | Uint8 pointer | Width x Count | The items array |
| keys | Uint8 pointer | 0x80 x Count | The key array |
| count | Uint32 | 32 | The item count |
| width | Uint32 | 32 | The maximum item width |

Table 13.1a the collection state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_COLLECTION\_KEY\_WIDTH | 16 | The key byte length |

Table 13.1b the collection constants

**API:**

**Collection Add**

Add an item to the collection. Pass in the initialized state, the item array pointer, and the pointer to the key to be associated with this item.

void qsc\_collection\_add(qsc\_collection\_state\* ctx, const uint8\_t\* item, const uint8\_t\* key)

**Collection Deserialize**

Convert a serialized collection stream into a collection state.

void qsc\_collection\_deserialize(qsc\_collection\_state\* ctx, const uint8\_t\* input)

**Collection Dispose**

Destroy the state and release the memory of a collection state.

void qsc\_collection\_dispose(qsc\_collection\_state\* ctx)

**Collection Erase**

Erase the collection and set all state members to zero.

void qsc\_collection\_erase(qsc\_collection\_state\* ctx)

**Collection Initialize**

Initialize the connection state. The width parameter defines the maximum width of an item.

void qsc\_collection\_initialize(qsc\_collection\_state\* ctx, size\_t width)

**Collection Item Exists**

Check if the item associated with the key is in the collection. Returns true if the item was found.

bool qsc\_collection\_item\_exists(const qsc\_collection\_state\* ctx, const uint8\_t\* key)

**Collection Item Find**

Find an item associated with a key in the collection, and copy it to the item parameter. Returns true if the item was found.

bool qsc\_collection\_find(const qsc\_collection\_state\* ctx, uint8\_t\* item, const uint8\_t\* key)

**Collection Item**

Retrieve a copy of a collection item from its index number. The index allows iteration through the internal items. The item at the index is copied to the item parameter.

void qsc\_collection\_item(qsc\_collection\_state\* ctx, uint8\_t\* item , size\_t index)

**Collection Item Remove**

Remove an item from the collection. The key is used to find the item, and the key and item are removed.

void qsc\_collection\_remove(qsc\_collection\_state\* ctx, const uint8\_t\* key)

**Collection Serialize**

Serialize the collection state to an array. The collection state is serialized to the output array. The size of the output array can be determined using the collection size call.

void qsc\_collection\_serialize(uint8\_t\* output, const qsc\_collection\_state\* ctx)

**Collection Size**

Returns the size of the serialized collection in its current state.

size\_t qsc\_collection\_size(const qsc\_collection\_state\* ctx)

**13.2 List**

**Header:**

list.h

**Description:**

A dynamic items list implementation.

**Structures:**

The **qsc\_list\_state** structure

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| items | Uint8 pointer | Width x Count | The items array |
| count | Uint64 | 64 | The item count |
| width | Uint64 | 64 | The maximum item width |

Table 12.2 the collection state structure

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_LIST\_ALIGNMENT | 64 | The list memory alignment |
| QSC\_LIST\_MAX\_DEPTH | 102400 | The maximum item count |

**API:**

**List Add**

Add an item to the list. Pass in the initialized state, and the item pointer.

void qsc\_list\_add(qsc\_list\_state\* ctx, void\* item)

**List Copy**

Copy an item from the list using the items index position. Pass in the initialized state, the index position, and the output item pointer.

void qsc\_list\_copy(qsc\_list\_state\* ctx, size\_t index, void\* item)

**List Count**

Get the number of items in the list.

size\_t qsc\_list\_count(const qsc\_list\_state\* ctx)

**List Deserialize**

Deserialize a list state stream, and populate a list state. Pass the empty list state and the pointer to the input stream.

void qsc\_list\_deserialize(qsc\_list\_state\* ctx, const uint8\_t\* input)

**List Deserialize**

Deserialize a list state stream, and populate a list state. Pass the empty list state and the pointer to the input stream.

void qsc\_list\_deserialize(qsc\_list\_state\* ctx, const uint8\_t\* input)

**List Dispose**

Erase the state and realease memory used by the list.

void qsc\_list\_dispose(qsc\_list\_state\* ctx)

**List Initialize**

Initialize the list state. Pass the empty state and the byte-size width of each item.

void qsc\_list\_initialize(qsc\_list\_state\* ctx, size\_t width)

**List Empty**

Returns true if there are no items in the array.

bool qsc\_list\_empty(const qsc\_list\_state\* ctx)

**List Full**

Returns true if the list has reached the maximum number of items.

void qsc\_list\_full(const qsc\_list\_state\* ctx)

**List Item**

Retrieve a copy of a list item from its index number. The index allows iteration through the internal items. The item at the index is copied to the item parameter.

void qsc\_list\_item(const qsc\_list\_state\* ctx, uint8\_t\* item, size\_t index)

**List Item**

Retrieve a copy of a list item from its index number. The index allows iteration through the internal items. The item at the index is copied to the item parameter.

void qsc\_list\_item(const qsc\_list\_state\* ctx, uint8\_t\* item, size\_t index)

**List Item Remove**

Remove an item from the list at the specified index. Pass the initialized list state and the index of the item to remove.

void qsc\_collection\_remove(qsc\_collection\_state\* ctx, size\_t index)

**List Serialize**

Serialize the list state to an array. The list state is serialized to the output array. The size of the output array can be determined using the list size call.

void qsc\_list\_serialize(qsc\_collection\_state\* ctx, uint8\_t\* output)

**List Size**

Returns the size of the serialized list in its current state.

size\_t qsc\_list\_size(const qsc\_list\_state\* ctx)

**List Sort**

Sort the items in the list.

size\_t qsc\_list\_sort(qsc\_list\_state\* ctx)

**13.3** **QSORT**

**Header:**

qsort.h

**Description:**

An implementation of QSORT sorting functions.

**Sort i8**

Sort an array of signed 8-bit integers by numerical value. Pass in the array to be sorted, and the start and end indexes within the array to sort.

size\_t qsc\_qsort\_sort\_i8(int8\_t\* arr8, int start, int end)

**Sort i16**

Sort an array of signed 16-bit integers by numerical value. Pass in the array to be sorted, and the start and end indexes within the array to sort.

size\_t qsc\_qsort\_sort\_i16(int8\_t\* arr16, int start, int end)

**Sort i32**

Sort an array of signed 32-bit integers by numerical value. Pass in the array to be sorted, and the start and end indexes within the array to sort.

size\_t qsc\_qsort\_sort\_i32(int8\_t\* arr32, int start, int end)

**Sort i64**

Sort an array of signed 64-bit integers by numerical value. Pass in the array to be sorted, and the start and end indexes within the array to sort.

size\_t qsc\_qsort\_sort\_i64(int8\_t\* arr64, int64\_t start, int64\_t end)

## **14:** **Integer and String Tools**

**14.1** **Array Utilities**

**Header:**

arrayutils.h

**Description:**

Array utilities; supporting string to integer functions

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_ARRAYTILS\_NPOS | -1 | The find string not found return value. |

Table 13.1a socket state constants

**API:**

**Find String**

Find the first instance of a token in a string, and return the char position. Takes the string, the search token, and the string length as parameters, and returns the tokens position within the string.

size\_t qsc\_arrayutils\_find\_string(const char\* str, size\_t slen, const char\* token)

**Hex To Uint8**

Converts a hexadecimal encoded string to a byte value. Takes the string, and the string length as parameters, and returns the byte value.

uint8\_t qsc\_arrayutils\_hex\_to\_uint8(const char\* str, size\_t slen)

**Uint8 To Hex**

Converts a byte value to hexadecimal and writes to a string. Takes a pointer to the char output array, the output length, and the value as parameters.

void qsc\_arrayutils\_uint8\_to\_hex(char\* output, size\_t outlen, uint8\_t value)

**String To Uint8**

Parse a 8-bit unsigned integer from a string. Takes a pointer to a char array, and the array length as parameters, and returns an 8-bit unsigned integer.

uint8\_t qsc\_arrayutils\_string\_to\_uint8(const char\* str, size\_t slen)

**String To Uint16**

Parse a 16-bit unsigned integer from a string. Takes a pointer to a char array, and the array length as parameters, and returns a 16-bit unsigned integer.

uint16\_t qsc\_arrayutils\_string\_to\_uint16(const char\* str, size\_t slen)

**String To Uint32**

Parse a 32-bit unsigned integer from a string. Takes a pointer to a char array, and the array length as parameters, and returns a 32-bit unsigned integer.

uint32\_t qsc\_arrayutils\_string\_to\_uint32(const char\* str, size\_t slen)

**String To Uint64**

Parse a 64-bit unsigned integer from a string. Takes a pointer to a char array, and the array length as parameters, and returns a 64-bit unsigned integer.

uint64\_t qsc\_arrayutils\_string\_to\_uint64(const char\* str, size\_t slen)

**14.2** **Console Utilities**

**Header:**

consoleutils.h

**Description:**

The console utilities class contains function used in creating inter-active console-based tools.

**Enumerations:**

The **qsc\_console\_font\_color** enumeration.

|  |  |
| --- | --- |
| Enumeration | Purpose |
| white | White font displayed on the console. |
| blue | Blue font displayed on the console. |
| green | Green font displayed on the console. |
| red | Red font displayed on the console. |

Table 13.2a font color enumeration

The **qsc\_console\_font\_style** enumeration

|  |  |
| --- | --- |
| Enumeration | Purpose |
| regular | Regular font displayed on the console. |
| bold | Bold font displayed on the console. |
| italic | Italic font displayed on the console. |
| bolditalic | Bold and italic font displayed on the console. |

Table 13.2b font style enumeration

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_CONSOLE\_MAX\_LINE | 128 | The maximum length of a console line. |

Table 13.2c console constants

**API:**

**Colored Message**

Color a line of console text. Takes the message string, and a font color enumeral as parameters.

void qsc\_consoleutils\_colored\_message(const char\* message, qsc\_console\_font\_color color)

**Get Char**

A blocking wait that returns a single character from console input. Returns the character value.

char qsc\_consoleutils\_get\_char()

**Get Line**

A blocking wait that returns a string from console input after a carriage return is detected. Returns the character string. Returns the size of the string, and takes a pointer to a character array, and the maximum line length as parameters.

size\_t qsc\_consoleutils\_get\_line(char\* line, size\_t maxlen)

**Get Formatted Line**

A blocking wait that returns a string from console input after a carriage return is detected. Removes the new-line, null, and carriage return characters. Returns the size of the string, and takes a pointer to the output character array, and the maximum line length as parameters.

size\_t qsc\_consoleutils\_get\_formatted\_line(char\* line, size\_t maxlen)

**Get Wait**

Pause the console until user-input is detected.

void qsc\_consoleutils\_get\_wait()

**Hex To Bin**

Convert a hexadecimal character string to a character byte array. Takes a pointer to the input hex string, a pointer to the output byte array, and the number of characters to convert as parameters.

void qsc\_consoleutils\_hex\_to\_bin(const char\* hexstr, uint8\_t\* output, size\_t length)

**Line Contains**

Find a set of characters in a line of console text. Takes the input line, and the search token as parameters, and returns true if the line contains the token.

bool qsc\_consoleutils\_line\_contains(const char\* line, const char\* token)

**Masked Password**

Gets a password masked on the console screen. Takes a pointer to an output byte array, and the output array length as parameters, and returns the size of the password.

size\_t qsc\_consoleutils\_masked\_password(uint8\_t\* output, size\_t outlen)

**Message Confirm**

User confirmation that and action can continue (Y/N y/n). Takes a pointer to the message array as the parameter, and returns true/false.

bool qsc\_consoleutils\_message\_confirm(const char\* message)

**Print Hex**

Convert a byte array to a hexadecimal string and print to the console. Takes a pointer to the input byte array, the input array length, and the line length as parameters. The output will include new line characters inserted at line length.

void qsc\_consoleutils\_print\_hex(const uint8\_t\* input, size\_t inputlen, size\_t linelen)

**Print Formatted**

Print a string to the console, ignoring special characters. Takes the input message and the input length as parameters.

void qsc\_consoleutils\_print\_formatted(const char\* input, size\_t inputlen)

**Print Formatted Line**

Print a string to the console, ignoring special characters, and add a line break. Takes the input message and the input length as parameters.

void qsc\_consoleutils\_print\_formatted\_line(const char\* input, size\_t inputlen)

**Print Safe**

Prints an array of characters to the console. Takes the input message as a parameter.

void qsc\_consoleutils\_print\_safe(const char\* input)

**Print Line**

Print an array of characters to the console with a line break.

void qsc\_consoleutils\_print\_line(const char\* input)

**Print Concatenated Line**

Print a concatenated set of character arrays, to the console with a line break between each. Takes a pointer to a multi-dimensional array of strings, and the number of sub-arrays as parameters..

void qsc\_consoleutils\_print\_concatenated\_line(const char\*\* input, size\_t count)

**Print Uint**

Print a 32-bit unsigned integer to the console. Takes the integer to print as a parameter.

void qsc\_consoleutils\_print\_uint(uint32\_t digit)

**Print Ulong**

Print an unsigned 64-bit integer to the console. Takes the integer to print as a parameter.

void qsc\_consoleutils\_print\_ulong(uint64\_t digit)

**Print Double**

Print a double floating point integer to the console. Takes the double to print as a parameter.

void qsc\_consoleutils\_print\_double(double digit)

**Progress Counter**

Prints a small spinning counter. Takes the duration in seconds as a parameter.

void qsc\_consoleutils\_progress\_counter(int32\_t seconds)

**Set Window Buffer**

Set the size of the window scroll buffers. Takes the desired horizontal and vertical sizes as parameters.

void qsc\_consoleutils\_set\_window\_buffer(size\_t width, size\_t height)

**Set Window Clear**

Clear the text from the window.

void qsc\_consoleutils\_set\_window\_clear()

**Set Window Prompt**

Set the window prompt string.

void qsc\_consoleutils\_set\_window\_prompt(const char\* prompt)

**Set Window Size**

Set the initial size of the console window. Takes the desired window width and height as parameters.

void qsc\_consoleutils\_set\_window\_size(size\_t width, size\_t height)

**Set Window Title**

Set the window title string. Takes a pointer to the window title string as a parameter.

void qsc\_consoleutils\_set\_window\_title(const char\* title)

**Set Virtual Terminal**

Enable virtual terminal mode.

void qsc\_consoleutils\_set\_virtual\_terminal()

**14.3** **Encoding**

**Header:**

encoding.h

**Description:**

Integer base64 and RAD encoding utilities.

**API:**

**Base64 Decode**

Decodes a base64 string to a byte array. Takes a pointer to the output byte array and input character arrays, the output and input array lengths as parameters.

bool qsc\_encoding\_base64\_decode(uint8\_t\* output, size\_t outlen, const char\* input, size\_t inlen)

**Base64 Decoded Size**

Gets the expected size of an array required by decoding. Takes a pointer to the input character array, and the array length as parameters, and returns the expected size of the decoded string.

size\_t qsc\_encoding\_base64\_decoded\_size(const char\* input, size\_t length)

**Base64 Encode**

Encode a byte array to a base64 string. Takes pointers to the output character array, and input byte arrays, and the input and output array lengths as parameters.

void qsc\_encoding\_base64\_encode(char\* output, size\_t outlen, const uint8\_t\* input, size\_t inplen)

**Base64 Encoded Size**

Gets the expected size of an array required by encoding. Takes a pointer to the input character array, and the array length as parameters, and returns the expected size of the encoded string.

size\_t qsc\_encoding\_base64\_decoded\_size(const char\* input, size\_t length)

**Is Valid Char**

Tests if an encoded character is a valid base64 encoding. Takes the character value as a parameter, and returns true/false.

bool qsc\_encoding\_base64\_is\_valid\_char(char value)

**14.4** **File Utilities**

**Header:**

fileutils.h

**Description:**

File utilities; contains common file related functions.

**API:**

**Working Directory**

Get the working directory path. Takes a pointer to a character array that receives the path, and returns true if the directory is loaded, false if the path array is too small.

bool qsc\_filetools\_working\_directory(char\* path)

**File Exists**

Test to see if a file exists. Takes a pointer to the search path as a parameter, and returns true/false.

bool qsc\_filetools\_file\_exists(const char\* path)

**File Size**

Returns the files size in bytes. Takes a pointer to the file path as a parameter, and returns the file size.

size\_t qsc\_filetools\_file\_size(const char\* path)

**Get Line**

Reads a line of text from a formatted file. Takes a pointer to the output char array, a pointer to the output length, and a pointer to the file structure as parameters, and returns the line size, or zero.

int64\_t qsc\_filetools\_getline(char\*\* line, size\_t\* length, FILE\* fp)

**Get Directory**

Get the directory portion of a full file path. The directory receives the directory portion of the fpath. The function returns the length of the string.

size\_t qsc\_fileutils\_get\_directory(char\* directory, size\_t dirlen, const char\* fpath)

**Get Extension**

Get the file extension portion of a full file path. The directory receives the directory portion of the fpath. The function returns the length of the string.

size\_t qsc\_fileutils\_get\_name(char\* name, size\_t namelen, const char\* fpath)

**Get Name**

Get the file name portion of a full file path. The directory receives the directory portion of the fpath. The function returns the length of the string.

size\_t qsc\_fileutils\_get\_name(char\* name, size\_t namelen, const char\* fpath)

**Append To File**

Append an array of characters to a file. Writes new data to the end of a binary file. Takes a pointer to the file path and the input stream, and the length of the input as parameters, and returns true for success.

bool qsc\_filetools\_append\_to\_file(const char\* path, const char\* stream, size\_t length)

**Create File**

Create a new file. Takes a pointer to the full file path as a parameter, and returns true if the file is created.

bool qsc\_filetools\_create\_file(const char\* path)

**Copy File**

Create a copy of a file. Takes a pointer to the full file inpath, and the new file path as parameters., and returns true if the file is copied.

bool qsc\_fileutils\_file\_copy(const char\* inpath, const char\* outpath)

**Copy Object To File**

Copy an object to a file. Takes pointers to the file path, and the object to copy, and the object size as parameters, and returns true if the object is copied.

bool qsc\_filetools\_copy\_object\_to\_file(const char\* path, const void\* obj, size\_t length)

**Copy Stream To File**

Copy a character array to a file. Takes pointers to the file path, and the array to copy, and the array size as parameters, and returns true if the array is copied.

bool qsc\_filetools\_copy\_stream\_to\_file(const char\* path, const char\* stream, size\_t length)

**Copy File To Object**

Copy a file to an object. Takes a pointer to the file path, a pointer to the object, and the object size as parameters, and returns the size in bytes that were copied.

size\_t qsc\_filetools\_copy\_file\_to\_object(const char\* path, void\* obj, size\_t length)

**Copy File To Stream**

Copy elements from a file to a byte array. Takes a pointer to the file path, a pointer to the input array, and the array length as parameters, and returns the size in bytes that were copied.

size\_t qsc\_filetools\_copy\_file\_to\_stream(const char\* path, char\* stream, size\_t length)

**Delete File**

Delete a file. Takes a pointer to the file path as a parameter, and returns true on success.

bool qsc\_filetools\_delete\_file(const char\* path)

**Erase File**

Erase a files content. Takes a pointer to the file path as a parameter, and returns true on success.

bool qsc\_filetools\_erase\_file(const char\* path)

**List Files**

Lists the files in a directory and their attributes. The result string receives each file name, attribute, creation and modified time in a formatted string. The function returns the length of the string.

size\_t qsc\_fileutils\_list\_files(char\* result, size\_t reslen, const char\* directory)

**Read Line**

Read a line of text from a file. Takes a pointer to the file path, a pointer to the output buffer, and a line number as parameters, and returns the number of characters read.

size\_t qsc\_filetools\_read\_line(const char\* path, char\* buffer, size\_t buflen, size\_t linenum)

**Safe Read**

Read a number of characters from a file. Takes a pointer to the file path, the starting position, the output array, and the length of bytes to read. The function returns the number of bytes read..

size\_t qsc\_fileutils\_safe\_read(const char\* fpath, size\_t position, char\* output, size\_t length)

**Safe Write**

Write a number of characters to a file. Takes a pointer to the file path, the starting position, the output array, and the length of bytes to read. The function returns the number of bytes read..

size\_t qsc\_fileutils\_safe\_write(const char\* fpath, size\_t position, const char\* input, size\_t length)

**Seek To**

Set the file pointer position. Pass in the file pointer, and the position from the start of the file. The function returns true if successful.

bool qsc\_fileutils\_seekto(FILE\* fp, size\_t position)

**Truncate File**

Truncate a file to a specified length. Pass the file handle and the truncated length of the file. The function returns true if successful.

bool qsc\_fileutils\_truncate\_file(FILE\* fp, size\_t length)

**Valid Path**

Checks a file path for valid formatting. The function returns true if successful.

bool qsc\_fileutils\_valid\_path(const char\* fpath)

**File Write**

Writes a number of bytes to a file. Pass the input parameter and input length, the starting position in the file, and the file handle. The function returns the number of bytes written to the file.

size\_t qsc\_fileutils\_write(const char\* input, size\_t inplen, size\_t position, FILE\* fp)

**File Write Line**

Writes a line of text to a file. Pass the file path, the input parameter and input length. The function returns true if successful.

bool qsc\_fileutils\_write\_line(const char\* fpath, const char\* input, size\_t inplen)

**File Zeroise**

Erase a file and set it to zero bytes length. Pass the file path parameter.

void qsc\_fileutils\_zeroise(const char\* fpath)

**14.5** **Folder Utilities**

**Header:**

folderutils.h

**Description:**

Folder utilities; common folder support functions.

**Enumerations:**

The **qsc\_folderutils\_directories** enumeration

|  |  |
| --- | --- |
| Enumeration | Purpose |
| qsc\_folderutils\_directories\_user\_app\_data | User App Data directory. |
| qsc\_folderutils\_directories\_user\_desktop | User Desktop directory. |
| qsc\_folderutils\_directories\_user\_documents | User Documents directory. |
| qsc\_folderutils\_directories\_user\_downloads | User Downloads directory. |
| qsc\_folderutils\_directories\_user\_favourites | User Favourites directory |
| qsc\_folderutils\_directories\_user\_music | User Music directory |
| qsc\_folderutils\_directories\_user\_pictures | User Pictures directory |
| qsc\_folderutils\_directories\_user\_programs | User Programs directory |
| qsc\_folderutils\_directories\_user\_shortcuts | User Shortcuts directory |
| qsc\_folderutils\_directories\_user\_videos | User Video directory |

Table 13.5a common directories enumeration

**API:**

**Append Delimiter**

Append a delimiter to a directory path. Pass the path parameter into the function.

void qsc\_folderutils\_append\_delimiter(char path[QSC\_SYSTEM\_MAX\_PATH])

**Create Directory**

Create a new folder. Takes the directory path as a parameter, and returns true if the folder is created.

bool qsc\_folderutils\_create\_directory(const char path[QSC\_SYSTEM\_MAX\_PATH])

**Delete Folder**

Delete a folder. Takes the directory path as a parameter, and returns true if the folder is deleted.

bool qsc\_folderutils\_delete\_directory(const char path[QSC\_SYSTEM\_MAX\_PATH])

**Directory Exists**

Check if a folder exists. Takes the directory path as a parameter, and returns true if the folder exists.

bool qsc\_folderutils\_directory\_exists(const char path[QSC\_SYSTEM\_MAX\_PATH])

**Directory List**

Lists the sub directories in a directory and their attributes. The result string receives each directory name, creation and modified time in a formatted string. The function returns true if successful

bool qsc\_folderutils\_directory\_exists(const char path[QSC\_SYSTEM\_MAX\_PATH])

**Get Directory**

Get the full path to a special system folder. Takes a directory enumerator and a pointer to an output array as parameters.

void qsc\_folderutils\_get\_directory(qsc\_folderutils\_directories directory, char output[QSC\_SYSTEM\_MAX\_PATH])

**Directory Has Delimiter**

Checks if the directory path ends in a delimiter. The function returns true if the path has the delimiter.

bool qsc\_folderutils\_directory\_has\_delimiter(const char path[QSC\_SYSTEM\_MAX\_PATH])

**14.6** **Integer Utilities**

**Header:**

intutils.h

**Description:**

Integer utilities; supporting integer related functions.

**API:**

**Are Equal8**

Compares two 8-bit integers arrays for equality. Parameters are pointers to both arrays, and the number of bytes to compare. Returns true if the arrays are equal.

bool qsc\_intutils\_are\_equal8(const uint8\_t\* a, const uint8\_t\* b, size\_t length)

**Be8To16**

Convert an 8-bit integer array to a 16-bit big-endian integer. Takes the input byte array as a parameter, and returns a 16-bit unsigned integer.

uint16\_t qsc\_intutils\_be8to16(const uint8\_t\* input)

**Be8To32**

Convert an 8-bit integer array to a 32-bit big-endian integer. Takes the input byte array as a parameter, and returns a 32-bit unsigned integer.

uint32\_t qsc\_intutils\_be8to32(const uint8\_t\* input)

**Be8To64**

Convert an 8-bit integer array to a 64-bit big-endian integer. Takes the input byte array as a parameter, and returns a 64-bit unsigned integer.

uint64\_t qsc\_intutils\_be8to64(const uint8\_t\* input)

**Be16To8**

Convert a 16-bit big-endian integer to a 8-bit unsigned integer array. Takes a pointer to the output array, and the 16-bit value as parameters.

void qsc\_intutils\_be16to8(uint8\_t\* output, uint16\_t value)

**Be32To8**

Convert a 32-bit big-endian integer to a 8-bit unsigned integer array. Takes a pointer to the output array, and the 32-bit value as parameters.

void qsc\_intutils\_be32to8(uint8\_t\* output, uint32\_t value)

**Be64To8**

Convert a 64-bit big-endian integer to a 8-bit unsigned integer array. Takes a pointer to the output array, and the 64-bit value as parameters.

void qsc\_intutils\_be64to8(uint8\_t\* output, uint64\_t value)

**Be8Increment**

Increment an 8-bit integer array as a segmented big-endian integer. Takes a pointer to the output byte array, and the arrays length as parameters.

void qsc\_intutils\_be8increment(uint8\_t\* output, size\_t outlen)

**BSwap32**

Byte reverse an array of 32-bit integers. Takes pointers to the destination and source arrays, and the number of 32-bit integers to convert.

void qsc\_intutils\_bswap32(uint32\_t\* destination, const uint32\_t\* source, size\_t length)

**BSwap64**

Byte reverse an array of 64-bit integers. Takes pointers to the destination and source arrays, and the number of 64-bit integers to convert.

void qsc\_intutils\_bswap64(uint64\_t\* destination, const uint64\_t\* source, size\_t length)

**Clear8**

Set an unsigned 8-bit integer array to zeroes. Takes a pointer to the 8-bit unsigned integer array, and the number of bytes to clear as parameters.

void qsc\_intutils\_clear8(uint8\_t\* a, size\_t count)

**Clear16**

Set an unsigned 16-bit integer array to zeroes. Takes a pointer to the 16-bit integer array, and the number of integers to clear as parameters.

void qsc\_intutils\_clear16(uint16\_t\* a, size\_t count)

**Clear32**

Set an unsigned 32-bit integer array to zeroes. Takes a pointer to the 32-bit integer array, and the number of integers to clear as parameters.

void qsc\_intutils\_clear32(uint32\_t\* a, size\_t count)

**Clear64**

Set an unsigned 64-bit integer array to zeroes. Takes a pointer to the 64-bit integer array, and the number of integers to clear as parameters.

void qsc\_intutils\_clear64(uint64\_t\* a, size\_t count)

**CMov**

Constant-time conditional move function. Takes pointers to the destination and source arrays, the number of bytes to move, and the condition as parameters.

void qsc\_intutils\_cmov(uint8\_t\* dest, const uint8\_t\* source, size\_t length, uint8\_t cond)

**Expand Mask**

Expand an integer mask in constant time. Takes the value, and returns the mask value.

size\_t qsc\_intutils\_expand\_mask(size\_t x)

**Are Equal**

Check if an integer is equal to a second integer. Takes the two integer values as parameters, and returns true if they are equal.

bool qsc\_intutils\_are\_equal(size\_t x, size\_t y)

**IsGte**

Check if an integer (x) is greater or equal to a second integer (y). Takes the integers as parameters, and returns true if x is greater or equal to y.

bool qsc\_intutils\_is\_gte(size\_t x, size\_t y)

**Hex To Bin**

Convert a hex string to an array. Takes pointers to the input hex string and output arrays, and the number of characters to process.

void qsc\_intutils\_hex\_to\_bin(const char\* hexstr, uint8\_t\* output, size\_t length)

**Bin To Hex**

Convert an array to a hex string. Takes pointers to the input byte array, and output hex character arrays, and the number of bytes to convert as parameters.

void qsc\_intutils\_bin\_to\_hex(const uint8\_t\* input, char\* hexstr, size\_t length)

**Le8Increment**

Increment an 8-bit integer array as a segmented little-endian integer. Takes a pointer to the output byte array, and the array’s length as parameters.

void qsc\_intutils\_le8increment(uint8\_t\* output, size\_t outlen)

**Le8Increment\_x128**

Increment the low 64-bit integer of a little-endian array by one. Takes a pointer to the 128-bit integer counter.

void qsc\_intutils\_leincrement\_x128(\_\_m128i\* counter)

**Le8Increment\_x512**

Offset increment the low 64-bit integer of a set of 64-bit pairs of a little-endian integers. Takes a pointer to the 512-bit integer counter.

void qsc\_intutils\_leincrement\_x512(\_\_m512i\* counter)

**Le8To16**

Convert an 8-bit integer array to a 16-bit little-endian integer. Takes the input byte array as a parameter, and returns a 16-bit unsigned integer.

uint16\_t qsc\_intutils\_le8to16(const uint8\_t\* input)

**Le8To32**

Convert an 8-bit integer array to a 32-bit little-endian integer. Takes the input byte array as a parameter, and returns a 32-bit unsigned integer.

uint32\_t qsc\_intutils\_le8to32(const uint8\_t\* input)

**Le8To64**

Convert an 8-bit integer array to a 64-bit little-endian integer. Takes the input byte array as a parameter, and returns a 64-bit unsigned integer.

uint64\_t qsc\_intutils\_le8to64(const uint8\_t\* input)

**Le16To8**

Convert a 16-bit little-endian integer to an 8-bit unsigned integer array. Takes a pointer to the output array, and the 16-bit value as parameters.

void qsc\_intutils\_be16to8(uint8\_t\* output, uint16\_t value)

**Le32To8**

Convert a 32-bit little-endian integer to a 8-bit unsigned integer array. Takes a pointer to the output array, and the 32-bit value as parameters.

void qsc\_intutils\_be32to8(uint8\_t\* output, uint32\_t value)

**Le64To8**

Convert a 64-bit integer to a little-endian 8-bit unsigned integer array. Takes a pointer to the output array, and the 64-bit value as parameters.

void qsc\_intutils\_b64to8(uint8\_t\* output, uint64\_t value)

**Max**

Return the larger of two integers. Takes the two comparison integers as parameters, and returns the largest value.

size\_t qsc\_intutils\_max(size\_t a, size\_t b)

**Min**

Return the smaller of two integers. Takes the two comparison integers as parameters, and returns the smallest value.

size\_t qsc\_intutils\_min(size\_t a, size\_t b)

**Reverse Bytes X128**

Reverse a 128-bit integer. Takes the input and output integers as parameters.

void qsc\_intutils\_reverse\_bytes\_x128(const \_\_m128i\* input, \_\_m128i\* output)

**Reverse Bytes X512**

Reverse a 512-bit integer. Takes the input and output integers as parameters.

void qsc\_intutils\_reverse\_bytes\_x512(const \_\_m512i\* input, \_\_m512i\* output)

**RotL32**

Rotate an unsigned 32-bit integer to the left. Takes the value and the number of bits to rotate that value as parameters, and returns the rotated integer.

uint32\_t qsc\_intutils\_rotl32(uint32\_t value, size\_t shift)

**RotL64**

Rotate an unsigned 64-bit integer to the left. Takes the value and the number of bits to rotate that value as parameters, and returns the rotated integer.

uint64\_t qsc\_intutils\_rotl64(uint64\_t value, size\_t shift)

**RotR32**

Rotate an unsigned 32-bit integer to the right. Takes the value and the number of bits to rotate that value as parameters, and returns the rotated integer.

uint32\_t qsc\_intutils\_rotr32(uint32\_t value, size\_t shift)

**RotR64**

Rotate an unsigned 64-bit integer to the right. Takes the value and the number of bits to rotate that value as parameters, and returns the rotated integer.

uint64\_t qsc\_intutils\_rotr64(uint64\_t value, size\_t shift)

**Verify**

Constant time comparison of two arrays of unsigned 8-bit integers. Takes pointers to the two arrays, and the number of bytes to compare as parameters. Returns zero in the arrays are equivalent.

int32\_t qsc\_intutils\_verify(const uint8\_t\* a, const uint8\_t\* b, size\_t length)

**14.7** **String Utilities**

**Header:**

stringutils.h

**Description:**

String utilities; common string support functions.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_STRINGUTILS\_TOKEN\_NOT\_FOUND | -1 | The search token was not found. |
| QSC\_STRINGUTILS\_HEX\_EXTENSION\_SIZE | 2 | The char size of the hex extension |
| QSC\_STRINGUTILS\_HEX\_BYTE\_SIZE | 2 | The char size of a hexidecimal byte |

Table 13.7a socket state constants

**API:**

**Formatting Count**

Counts all whitespaces, line stops, and return characters from a string. Takes a pointer to the destination string, and the destination length as parameters, and returns the new string length.

size\_t qsc\_stringutils\_formatting\_count(const char\* dest, size\_t dstlen)

**Formatting Filter**

Remove all whitespaces, line stops, and return characters from a string. Takes pointers to the source and destination strings, and the source length as parameters, and returns the size of the new string.

size\_t qsc\_stringutils\_formatting\_filter(const char\* source, size\_t srclen, char\* filtered)

**Add Line Breaks**

Add line breaks to a string at a line length interval. Takes pointers to the destination buffer and the source string, the destination length, the line length, and the source length as parameters, returns the size of the new string.

size\_t qsc\_stringutils\_add\_line\_breaks(char\* dest, size\_t dstlen, size\_t linelen, const char\* source, size\_t srclen)

**Remove Line Breaks**

Removes all line breaks from a string. Takes pointers to the destination buffer and source strings, and the destination length and source length as parameters, returns the size of the new string.

size\_t qsc\_stringutils\_remove\_line\_breaks(char\* dest, size\_t dstlen, const char\* source, size\_t srclen)

**Clear String**

Clear a string of data. Takes a pointer to the source array as a parameter.

void qsc\_stringutils\_clear\_string(char\* source)

**Clear SubString**

Clear a length of data from a string. Takes a pointer to the source array and the number of characters to clear, as parameters.

void qsc\_stringutils\_clear\_substring(char\* dest, size\_t length)

**Compare Strings**

Compare two strings for equivalence. Takes pointers for the two strings to compare, and the number of characters to compare as parameters, returns true if the arrays are equal.

bool qsc\_stringutils\_compare\_strings(const char\* str1, const char\* str2, size\_t length)

**Concat Strings**

Concatenate two strings. Takes the pointers to the destination buffer and input strings, and the destination length as parameters, and returns the size of the new string.

size\_t qsc\_stringutils\_concat\_strings(char\* dest, size\_t dstlen, const char\* source)

**Concat and Copy**

Concatenate two strings and copy them to a third string. Takes the pointers to the destination buffer, the first and second substrings, and the destination length parameters, and returns the length of the new string.

size\_t qsc\_stringutils\_concat\_and\_copy(char\* dest, size\_t dstlen, const char\* str1, const char\* str2)

**Copy SubString**

Copy a length of one string to another. Takes pointers to the destination buffer and source string, and the destination and source lengths as parameters, and returns the size of the new strings.

size\_t qsc\_stringutils\_copy\_substring(char\* dest, size\_t dstlen, const char\* source, size\_t srclen)

**Copy String**

Copy a source string to a destination string. Takes pointers to the source and destination arrays, and the destination lengthe as parameters, and returns the new string size.

size\_t qsc\_stringutils\_copy\_string(char\* dest, size\_t dstlen, const char\* source)

**Find Char**

Find a character position within a string. Takes a pointer to the source array, and the search token string as parameters, and returns the token position withing the source or -1 if the token is not found.

int64\_t qsc\_stringutils\_find\_char(const char\* source, const char token)

**Find String**

Find a substrings position within a string. Takes a pointer to the source array, and the search token string as parameters, and returns the token position withing the source or -1 if the token is not found.

int32\_t qsc\_stringutils\_find\_string(const char\* source, const char\* token)

**Byte To Hex**

Convert a byte to a hexidecimal string.

int32\_t qsc\_stringutils\_find\_string(const char\* source, const char\* token)

**Hex To Byte**

Convert a hexidecimal string to a byte.

uint8\_t qsc\_stringutils\_hex\_to\_byte(const char\* hex)

**Insert String**

Inserts a substring into a string. Takes pointers to the destination and souce strings, the destination length, and the insert offset as parameters, returns the new string length or -1 on failure.

int32\_t qsc\_stringutils\_insert\_string(char\* dest, size\_t dstlen, const char\* source, size\_t offset)

**Is Alpha Numeric**

Check that a string contains only alpha numeric ascii characters. Takes the source and source length as parameters, and returns true if the string is alpha-numeric.

bool qsc\_stringutils\_is\_alpha\_numeric(const char\* source, size\_t srclen)

**Is Empty**

Check if a string contains and characters.

bool qsc\_stringutils\_is\_empty(const char\* source)

**Is Hex**

Check that a string contains only hexadecimal ascii characters. Takes the source and source length as parameters, and returns true if the string is in hexadecimal.

bool qsc\_stringutils\_is\_hex(const char\* source, size\_t srclen)

**Is Numeric**

Check that a string contains only numeric ascii characters. Takes the source and source length as parameters, and returns true if the string is in hexadecimal.

bool qsc\_stringutils\_is\_numeric(const char\* source, size\_t srclen)

**Join String**

Join an array of strings to form one string. Takes an array of sub-strings and the array count as parameters, and returns the joined string.

char\* qsc\_stringutils\_join\_string(char\*\* source, size\_t count)

**Register String**

Join an array of strings to form one string.

char\* qsc\_stringutils\_register\_string(char\*\* source, size\_t count)

**Remove Null Chars**

Remove null characters from an array.

size\_t qsc\_stringutils\_remove\_null\_chars(char\* source, size\_t srclen)

**Reverse Find String**

Find the position of a substring within a string, searching in reverse.

int64\_t qsc\_stringutils\_reverse\_find\_string(const char\* source, const char\* token, size\_t start)

**Reverse SubString**

Find a substring within a string, searching in reverse. Takes a pointer to the source string, and the search token as parameters, and returns the sub-string.

char\* qsc\_stringutils\_reverse\_sub\_string(const char\* source, const char\* token)

**String Contains**

Test if the string contains a substring. Takes a pointer to the source string, and the search token as parameters, and returns true if the source contains token.

bool qsc\_stringutils\_string\_contains(const char\* source, const char\* token)

**Strings Equal**

Compare two strings for equality.

bool qsc\_stringutils\_strings\_equal(const char\* str1, const char\* str2)

**Split String**

Split a string into a substring 2-dimensional array. Takes pointers to the source array, the delimiter, and the sub-string count, and returns the split string, or NULL on failure.

char\*\* qsc\_stringutils\_split\_string(char\* source, const char\* delim, size\_t\* count)

**Split Strings**

Split a string into two substrings.

void qsc\_stringutils\_split\_strings(char\* dest1, char\* dest2, size\_t destlen, const char\* source, const char\* token)

**Sub String**

Find a substring within a string. Takes a pointer to the source string, and the search token as parameters, and returns a pointer to the substring.

char\* qsc\_stringutils\_sub\_string(const char\* source, const char\* token)

**String To Int**

Convert a string to a 32-bit integer. Takes a pointer to the source string as a parameter, and returns a 32-bit signed integer.

int32\_t qsc\_stringutils\_string\_to\_int(const char\* source)

**String Size**

Get the character length of a string. Takes a pointer to the source string as a parameter, and returns the string length.

size\_t qsc\_stringutils\_string\_size(const char\* source)

**Int To String**

Convert a 32-bit signed integer to a string. Takes the integer, a pointer to the destination string, and the destination length as parameters.

void qsc\_stringutils\_int\_to\_string(int32\_t num, char\* dest, size\_t dstlen)

**UInt32 To String**

Convert a 32-bit unsigned integer to a string.

void qsc\_stringutils\_uint32\_to\_string(uint32\_t num, char\* dest, size\_t destlen)

**Int64 To String**

Convert a 64-bit signed integer to a string.

void qsc\_stringutils\_int64\_to\_string(int64\_t num, char\* dest, size\_t destlen)

**UInt64 To String**

Convert a 64-bit unsigned integer to a string.

void qsc\_stringutils\_uint64\_to\_string(uint64\_t num, char\* dest, size\_t destlen)

**To Lowercase**

Convert a string to all lower -characters. Takes a pointer to the source string as a parameter.

void qsc\_stringutils\_to\_lowercase(char\* source)

**To Uppercase**

Convert a string to all upper-case characters. Takes a pointer to the source string as a parameter.

void qsc\_stringutils\_to\_uppercase(char\* source)

**Trim Newline**

Trim null and newline characters from a string. Takes a pointer to the source string as a parameter.

void qsc\_stringutils\_trim\_newline(char\* source)

**Trim Spaces**

Trim null and newline characters from a string. Takes a pointer to the source string as a parameter.

void qsc\_stringutils\_trim\_spaces(char\* source)

**Whitespace Count**

Count all the whitespaces in a string. Takes a pointer to the source string, and the source length as parameters.

size\_t qsc\_stringutils\_whitespace\_count(const char\* source, size\_t srclen)

**Whitespace Filter**

Remove all the whitespaces from a string. Takes pointers to the source and destination strings, and the source length as parameters.

size\_t qsc\_stringutils\_whitespace\_filter(const char\* source, size\_t srclen, char\* filtered)

**14.8** **System Utilities**

**Header:**

sysutils.h

**Description:**

System specific functions; provides system specific statistics, counters, and feature availability information.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_SYSUTILS\_SYSTEM\_NAME\_MAX | 256 | The system maximum name length. |

Table 13.18 system constants

**Structures:**

The **qsc\_sysutils\_drive\_space\_state** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| free | Uint64 | 0x40 | The free drive space. |
| total | Uint64 | 0x40 | The total drive space. |
| avail | Uint64 | 0x40 | The available drive space. |

Table 13.18b drive space structure

The **qsc\_sysutils\_memory\_statistics\_state** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| phystotal | Uint64 | 0x40 | The total physical memory. |
| physavail | Uint64 | 0x40 | The available physical memory. |
| virttotal | Uint64 | 0x40 | The total virtual memory. |
| virtavail | Uint64 | 0x40 | The available virtual memory. |

Table 13.18b drive space structure

**API:**

**Computer Name**

Get the computer string name. Takes a pointer to the name string buffer as a parameter, and returns the length of the computer name.

size\_t qsc\_sysutils\_computer\_name(char\* name)

**Drive Space**

Get the system drive space statistics. Takes a pointer to the drive string, and the drive space state structure as parameters.

void qsc\_sysutils\_drive\_space(const char\* drive, qsc\_sysutils\_drive\_space\_state\* state)

**RdRand Available**

Check if the system supports Intel RDRAND random seed generator. Returns true if the service is available on the system.

bool qsc\_sysutils\_rdrand\_available()

**RdSeed Available**

Check if the system supports Intel RDSEED random seed generator. Returns true if the service is available on the system.

bool qsc\_sysutils\_rdseed\_available()

**RdSeed Available**

Check if the system supports Intel RDSEED random seed generator. Returns true if the service is available on the system.

bool qsc\_sysutils\_rdseed\_available()

**RdTsc Available**

Check if the system has a high resolution RDTSC timer.

bool qsc\_sysutils\_rdtsc\_available()

**Memory Statistics**

Get the memory statistics from the system. Takes a pointer to the memory statistics state as a parameter.

void qsc\_sysutils\_memory\_statistics(qsc\_sysutils\_memory\_statistics\_state\* state)

**Process Id**

Get the current process id. Returns the process id.

uint32\_t qsc\_sysutils\_process\_id()

**User Name**

Get the systems logged-on user name string. Takes the name buffer string as a parameter.

size\_t qsc\_sysutils\_user\_name(char\* name)

**System Uptime**

Get the system uptime since boot.

uint64\_t qsc\_sysutils\_system\_uptime()

**System Timestamp**

Get the current high-resolution time-stamp. Returns the 64-bit timestamp.

uint64\_t qsc\_sysutils\_system\_timestamp()

**User Identity**

Get the users identity string. Takes pointers to the user-name string, and the output id strings as parameters.

void qsc\_sysutils\_user\_identity(const char\* name, char\* id)

**14.9** **TimerEx**

**Header:**

timerex.h

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_TIMEREX\_TIMESTAMP\_MAX | 80 | The timestamp maximum byte size. |

**Description:**

Timer functions.

**Get Date**

Get the calendar date from the current locale. Writes the time string to the output pameter.

void qsc\_timerex\_get\_date(char output[QSC\_TIMEREX\_TIMESTAMP\_MAX]))

**Get DateTime**

Get the calendar date and the time from the current locale. Writes the time string to the output pameter.

void qsc\_timerex\_get\_datetime(char output[QSC\_TIMEREX\_TIMESTAMP\_MAX]))

**Get Time**

Get the time from the current locale. Writes the time string to the output pameter.

void qsc\_timerex\_get\_time(char output[QSC\_TIMEREX\_TIMESTAMP\_MAX]))

**Stopwatch Start**

Returns the clock time at the start of a timed operation. Returns a clock\_t time structure.

clock\_t qsc\_timerex\_stopwatch\_start()

**StopWatch Elapsed**

Returns the time difference between the start and current time in milliseconds. Takes a clock\_t structure as a parameter, and returns the elapsed milliseconds.

uint64\_t qsc\_timerex\_stopwatch\_elapsed(clock\_t start)

**14.10** **Time Stamp**

**Header:**

timestamp.h

**Description:**

Timestamp function definitions.

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_TIMESTAMP\_EPOCH\_START | 1900 | The system maximum name length. |
| QSC\_TIMESTAMP\_STRING\_SIZE | 20 | The size of the timestamp string. |
| QSC\_TIMESTAMP\_SECONDS\_PER\_MINUTE | 60 | The seconds in a minute. |
| QSC\_TIMESTAMP\_SECONDS\_PER\_HOUR | 3600 | The seconds in an hour. |
| QSC\_TIMESTAMP\_SECONDS\_PER\_DAY | 86400 | The seconds in a day. |

Table 14.10 timestamp constants

**API:**

**Current Date**

Get the calendar date from the current locale. Takes the output string as a parameter.

void qsc\_timestamp\_current\_date(char output[QSC\_TIMESTAMP\_STRING\_SIZE])

**Current DateTime**

Get the calendar date and time from the current locale. Timestamp string format is YYYY-MM-DD HH-MM-SS. Takes the output string as a parameter.

void qsc\_timestamp\_current\_datetime(char output[QSC\_TIMESTAMP\_STRING\_SIZE])

**Current Time**

Get the local time. Takes the output string as a parameter.

void qsc\_timestamp\_current\_time(char output[QSC\_TIMESTAMP\_STRING\_SIZE])

**EpochTime Seconds**

Get the date and time from the current locale in seconds from epoch. Returns the 64-bit seconds since epoch.

uint64\_t qsc\_timestamp\_epochtime\_seconds()

**Time Struct To String**

Convert a time structure to a date and time string. Timestamp string format is YYYY-MM-DD HH-MM-SS. Takes the output string and a pointer to the time struct as parameters.

void qsc\_timestamp\_time\_struct\_to\_string(char output[QSC\_TIMESTAMP\_STRING\_SIZE], const struct tm\* tstruct)

**String To Time Struct**

Convert a date and time string to a time structure. Timestamp string format must be YYYY-MM-DD HH-MM-SS. Takes the output string and a pointer to the time struct as parameters.

void qsc\_timestamp\_string\_to\_time\_struct(struct tm\* tstruct, const char input[QSC\_TIMESTAMP\_STRING\_SIZE])

**DateTime Seconds Remaining**

Compare a base datetime with another future datetime string, and return the difference in seconds. Takes the base-time string, and the comparison string as parameters, and returns the number of seconds in the date-time string.

uint64\_t qsc\_timestamp\_datetime\_seconds\_remaining(const char basetime[QSC\_TIMESTAMP\_STRING\_SIZE], const char comptime[QSC\_TIMESTAMP\_STRING\_SIZE])

**DateTime To Seconds**

Convert the date-time string to a seconds from epoch unsigned 64-bit integer. Pass the datetime string in the input parameter, retiurns the time in seconds..

uint64\_t qsc\_timestamp\_datetime\_to\_seconds (const char input[QSC\_TIMESTAMP\_STRING\_SIZE])

**Seconds To DateTime**

Convert a second’s count from epoch-time to a date-time string. Takes the number of seconds between the clock epoch time and now, and the output time string as parameters.

void qsc\_timestamp\_seconds\_to\_datetime(uint64\_t tsec, char output[QSC\_TIMESTAMP\_STRING\_SIZE])

**14.11** **Transpose**

**Header:**

transpose.h

**Description:**

String and array transposition functions.

**API:**

**Bytes To Native**

Convert 32-bit integers in big-endian format to 8-bit integers. Takes a pointer to the 32-bit integer output and 8-bit input arrays, and the length as parameters.

void qsc\_transpose\_bytes\_to\_native(uint32\_t\* output, const uint8\_t\* input, size\_t length)

**Hex To Bin**

Convert a hexadecimal string to a decimal 8-bit array. Takes pointers to the output and input arrays, and the length as parameters.

void qsc\_transpose\_hex\_to\_bin(uint8\_t\* output, const char\* input, size\_t length)

**Native To Bytes**

Convert 8-bit integers to 32-bit integers in big-endian format. Takes a pointer to the 8-bit integer output and32-bit input arrays, and the length as parameters.

void qsc\_transpose\_native\_to\_bytes(uint8\_t\* output, const uint32\_t\* input, size\_t length)

**String To Scalar**

Convert an 8-bit character array to zero padded 32-bit scalar integers. Takes a pointer to the 32-bit integer output and 8-bit input arrays, and the length as parameters.

void qsc\_transpose\_string\_to\_scalar(uint32\_t\* output, const char\* input, size\_t count)

**14.12** **WinUtils**

**Header:**

winutils.h

**Description:**

Windows operating system functions.

**Enumerations:**

The **qsc\_winutils\_registry\_value\_types** enumeration

|  |  |
| --- | --- |
| Enumeration | Purpose |
| REG\_SZ\_TYPE | String value type. |
| REG\_DWORD\_TYPE | DWORD value type. |
| REG\_QWORD\_TYPE | QWORD value type. |
| REG\_BINARY\_TYPE | Binary value type. |

Table 14.12a registry values enumeration

The **qsc\_winutils\_service\_states** enumeration

|  |  |
| --- | --- |
| Enumeration | Purpose |
| QSC\_WINUTILS\_SERVICE\_START | Start the service. |
| QSC\_WINUTILS\_SERVICE\_STOP | Stop the service.. |
| QSC\_WINUTILS\_SERVICE\_PAUSE | Pause the service. |
| QSC\_WINUTILS\_SERVICE\_RESUME | Resume the service. |

Table 14.12b service states enumeration

**Constants:**

|  |  |  |
| --- | --- | --- |
| Constant Name | Value | Purpose |
| QSC\_WINTOOLS\_ATTRIBUTES\_BUFFER\_SIZE | 256 | The file attributes buffer size. |
| QSC\_WINTOOLS\_NETSTAT\_BUFFER\_SIZE | 1024 | The network statistics buffer size. |
| QSC\_WINTOOLS\_NETSTAT\_NAME\_SIZE | 256 | The network statistics name size. |
| QSC\_WINTOOLS\_PROCESS\_LIST\_SIZE | 16384 | The process list buffer size. |
| QSC\_WINTOOLS\_REGISTRY\_BUFFER\_SIZE | 1024 | The registry buffer size. |
| QSC\_WINTOOLS\_REGISTRY\_LIST\_SIZE | 8192 | The registry list buffer size. |
| QSC\_WINTOOLS\_RUNAS\_BUFFER\_SIZE | 260 | The runas buffer size. |
| QSC\_WINTOOLS\_SERVICE\_LIST\_SIZE | 16384 | The service list buffer size |
| QSC\_WINTOOLS\_SERVICE\_BUFFER\_SIZE | 512 | The service buffer size. |
| QSC\_WINTOOLS\_SERVICE\_LIST\_DESCRIPTION | NA | Include the service descriptions in the service list output. |
| QSC\_WINTOOLS\_SERVICE\_LIST\_ACTIVE\_ONLY | NA | Only include running services when listing services |

Table 14.10 timestamp constants

**API:**

**File Get Attributes**

Get a string of file attributes. Takes the path string as a parameter, and outputs the string to the result array.

size\_t qsc\_winutils\_file\_get\_attributes(char\* result, size\_t reslen, const char\* path)

**File Set Attributes**

Set a file attribute. Takes the path string as a parameter, and the attribute string. Valid attributes are readonly, hidden, system, archive, normal, temporary, offline, noindex, encrypted.

bool qsc\_winutils\_file\_set\_attribute(const char\* path, const char\* attr)

**Network Statistics**

Get a list of network statistics including address and adaptor information. The result string receives the formatted printable list. The function returns the size of the list string.

size\_t qsc\_winutils\_network\_statistics(char\* result, size\_t reslen)

**Process List**

Get a list of running system processes. The result string receives the formatted printable list. The function returns the size of the list string.

size\_t qsc\_winutils\_process\_list(char\* result, size\_t reslen)

**Process Terminate**

Terminate a process. Pass the process name parameter. The function returns true if the process was terminated.

bool qsc\_winutils\_process\_terminate(const char\* name)

**Registry Value Add**

Add a value to a registry subkey. The keypath specifies the full string path including the root key. The value string contains the value, and the vtype specifies the type of value. The function returns true if successful.

bool qsc\_winutils\_registry\_key\_add(const char\* keypath, const char\* value, qsc\_winutils\_registry\_value\_types vtype)

**Registry Key Delete**

Delete a subkey and its values from the registry. The function returns true if successful.

bool qsc\_winutils\_registry\_key\_delete(const char\* keypath)

**Registry Key List**

Copy a list of subkeys under a registry key. The keypath contains the full path to the key including the root key. The result string receives the formatted printable list. The function returns the size of the list string.

size\_t qsc\_winutils\_registry\_key\_list(char\* result, size\_t reslen, const char\* keypath)

**Run Executable**

Run an executable. The path specifies the full path to the executable. The function returns true if successful.

bool qsc\_winutils\_run\_executable(const char\* path)

**Run As**

Run an executable under a user account. The expath specifies the full path to the executable. The user and password are the users credentials. The function returns true if successful.

bool qsc\_winutils\_run\_as\_user(const char\* user, const char\* password, const char\* expath)

**Service List**

Get a list of running system services. The result string receives the formatted printable list. The function returns the size of the list string.

size\_t qsc\_winutils\_service\_list(char\* result, size\_t reslen)

**Service List Size**

Get the service list size.

size\_t qsc\_winutils\_service\_list\_size()

**Service List State**

Change a service state. Pass the service name and the desired state in as parameters. The function returns true on success.

bool qsc\_winutils\_service\_state(const char\* name, qsc\_winutils\_service\_states state)

**User List**

Get a list of system user accounts. The result string receives the formatted printable list. The function returns the size of the list string.

size\_t qsc\_winutils\_user\_list(char\* result, size\_t reslen)

**User Current**

Get the name of the logged-in user account. The result string receives the list name and group. The function returns the size of the string.

size\_t qsc\_winutils\_current\_user(char\* result, size\_t reslen)

## **15:** **Memory and Processor**

**15.1** **CPUID**

**Header:**

cpuid.h

**Description:**

Contains the CPU feature availability functions.

**Structures:**

The **qsc\_cpu\_features** structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Name | Data Type | Bit Length | Function |
| aesni | Bool | 0x08 | The AESNI flag. |
| avx | Bool | 0x08 | The AVX flag. |
| avx2 | Bool | 0x08 | The AVX2 flag. |
| avx512 | Bool | 0x08 | The AVX512 flag. |
| hyperthread | Bool | 0x08 | The hyper thread flag. |
| pcmul | Bool | 0x08 | The PCMUL flag. |
| rdtcsp | Bool | 0x08 | The RDTCSP flag. |
| cacheline | Uint32 | 0x20 | The number of cache lines. |
| cores | Uint32 | 0x20 | The number of cores. |
| cpus | Uint32 | 0x20 | The number of CPUs. |
| freqbase | Uint32 | 0x20 | The frequency base. |
| l1cache | Uint32 | 0x20 | The L1 cache size. |
| l2cache | Uint32 | 0x20 | The L2 cache size. |
| serial | Char Array | 0x40 | The CPU serial number. |
| vendor | Char Array | 0x60 | The CPU vendor name. |

Table 14.1a cpu features structure

**API:**

**Runtime Features**

Get a list of supported CPU features. Takes a pointer to a CPU features structure as a parameter, and returns true for success.

bool qsc\_runtime\_features(qsc\_cpu\_features\* const features)

**15.2** **SIMD Memory Utilities**

**Header:**

memutils.h

**Description:**

Memory utilities; contains common memory related functions, implemented with AVX/AVX2/AVX512 SIMD instructions.

**API:**

**Prefetch L1**

Prefetch memory to L1 cache. Takes the output memory address and the number of bytes to cache as parameters.

void qsc\_memutils\_prefetch\_l1(uint8\_t\* address, size\_t length)

**Prefetch L2**

Prefetch memory to L2 cache. Takes the output memory address and the number of bytes to cache as parameters.

void qsc\_memutils\_prefetch\_l2(uint8\_t\* address, size\_t length)

**Prefetch L3**

Prefetch memory to L3 cache. Takes the output memory address and the number of bytes to cache as parameters.

void qsc\_memutils\_prefetch\_l3(uint8\_t\* address, size\_t length)

**Aligned Alloc**

Allocate an aligned 8-bit integer array. Takes the alignment size and the length of the array as parameters, and returns an aligned array.

void\* qsc\_memutils\_aligned\_alloc(int32\_t align, size\_t length)

**Aligned Free**

Free an aligned memory block created with aligned-alloc. Takes a pointer to an aligned array.

void qsc\_memutils\_aligned\_free(void\* block)

**Clear**

Erase a block of memory. Takes a pointer to the array, and the number of bytes to erase.

void qsc\_memutils\_clear(void\* output, size\_t length)

**Copy**

Copy a block of memory. Takes pointers to the output and input arrays, and the number of bytes to copy, as parameters.

void qsc\_memutils\_copy(void\* output, const void\* input, size\_t length)

**SetValue**

Set a block of memory to a value. Takes a pointer to an output array, the number of bytes to change, and the value as parameters.

void qsc\_memutils\_setvalue(void\* output, size\_t length, uint8\_t value)

**XOR**

Bitwise XOR two blocks of memory. Takes pointers to the output and input arrays, and the number of bytes to XOR as parameters.

void qsc\_memutils\_xor(uint8\_t\* output, const uint8\_t\* input, size\_t length)

**XORV**

Bitwise XOR a block of memory with a byte value. Takes a pointer to an output array, the number of bytes to XOR, and the value as parameters.

void qsc\_memutils\_xorv(uint8\_t\* output, const uint8\_t value, size\_t length)

**15.3** **Secure Memory Functions**

**Header:**

secmem.h

**Description:**

Secure memory functions; contains secure memory-locked functions.

**API:**

**Secmem Alloc**

Allocate a block of secure memory. Takes the number of bytes to allocate as a parameter, and returns a pointer to the allocated memory.

uint8\_t\* qsc\_secmem\_alloc(size\_t length)

**Secmem Release**

Erase a length of secure memory. Takes a pointer to the memory to erase. And the number of bytes to erase as parameters.

void qsc\_secmem\_erase(uint8\_t\* block, size\_t length)

**Secmem Free**

Erase and free a block of secure memory.

void qsc\_secmem\_free(uint8\_t\* block, size\_t length)

**Secmem Page Size**

Returns the internal memory page size. Returns the system memory page boundary size.

size\_t qsc\_secmem\_page\_size()

## **Annex A:** **Design Specifications**

**Design Objectives**

The QSC library was designed to be a lightweight, intuitive, and powerful set of cryptographic tools. We took many different considerations into account while designing this library; it had to be easy to use, so as to avoid implementation mistakes. It had to be compact, so as to fit on a large range of hardware implementations. The library has to be fast, to be competitive in high-performance environments. Most of all though, we have designed QSC to be powerful; this library contains some of the most advanced, and future-secure cryptographic primitives available in the world. We developed a library that will withstand the technological changes that are certain to come, as we venture into the quantum age.

We chose to make the library MISRA compliant, a very strict and comprehensive set of secure C programming-language guidelines. We upgraded existing codes to this format, including asymmetric ciphers and signature schemes, and have followed a strict regimen of secure programming practices throughout the library. Many industries now insist on code that conforms to these rule-sets, including the automobile industry, military and government organizations, and the aeronautics industry. Source codes must conform to these standards, or they will not be accepted by a rapidly growing number of industries. These rules were designed by experts in computer security, and MISRA has become the standard of computer-code conformance when using the C or C++ language in a secure context.

We have integrated SIMD instructions throughout the library, spanning many different supporting functions and cryptographic primitives. Most server hardware is now SIMD capable, and we have implemented those instructions as optional implementations for compatible CPUs. We have used the modern subsets of SIMD instructions; AVX, AVX2, and the newest version AVX512, in functions integrated into the library, from common memory functions, to entire primitives like Falcon, being implemented in SIMD instructions. The library also contains support for asynchronous threading, parallel loops, event callbacks, and a variety of multi-threading support functions.

We have implemented secure IPv4 and IPv6 networking stacks. These TCP/IP networking functions, cover a wide range of network operations, including synchronous and asynchronous sockets, complete IPv6 support, and parallel implementations of **Winsock** for Windows devices, or **Berkely** **Sockets** for Unix and Linux. We have added simplified server and client functionality, including asynchronous server *accept* and *receive* operations. We have also added a queuing implementation, and many peripheral network support operations, all in a straightforward, easy-to-use format.

A thorough testing framework is critical to developing secure code, and we have implemented a comprehensive set of testing functions, that perform operation wellness tests, known answer tests, and stress tests on every cryptographic primitive. We use only the official and most-current known answer tests (KATs), including the NIST PQ3 official tests for the asymmetric primitives. In addition to this are custom stress and wellness tests, and official testing frameworks like the NIST AESAVS tests, and the official NIST SHA3 tests.

The library organization and naming conventions take on a natural language approach, where the API and function purpose are synonymous, making the implementation as obvious as possible. We re-use many keywords throughout the library like *initialize*, *update*, and *generate*, to create a cohesive set of instruction meanings, and define a memorable repetition-based function language.

**API Naming Patterns**

* All public functions are prefixed with the library name **qsc\_**
* The API naming convention begins with the library-name prefix, followed by the primitive or function-family name, and ending with the function-name, ex.

**void qsc\_aes\_dispose(qsc\_aes\_state\* state)**

* Pointer types shall place the pointer character (\*) alongside the integer type ex.

**void qsc\_stringutils\_clear\_string(char\* source)**

* The function portion of a function name should not contain more than 3 words, and ideally, use only a single word, ex. **qsc\_kyber\_generate(…)**, the ‘generate’ portion is the function name, ‘kyber’ is the function-family name, and ‘qsc’ is the root library name
* A function state should always be the first parameter in a function call, even if it is declared a constant. The function parameters should always have output receiving parameters before constant parameters, but after a function state, ex.

**void qsc\_aes\_cbc\_encrypt\_block(qsc\_aes\_state\* state, uint8\_t\* output, const uint8\_t\* input)**

* A function that returns a binary value (0/1), shall use the **bool** integer type
* A function that returns from a range of condition values (-1/0/1), shall return an **enumeration** type
* A function that returns a length or size value, shall return a **size\_t** type
* Function variables should be declared at the top function, or at the top of an inner scope
* Function variables shall not be initialized in the declaration line
* Header files should only contain public functions, whenever possible all other functions should be declared static in the implementation file.
* Each primitives functions should be contained in a single implementation file whenever possible. Exceptions should be in files with the format *primitive-base*, ex. **kyberbase.h**
* To differentiate static functions in an implementation file, the library prefix should be excluded from static functions in the implementation file to denote locality, ex.

**static** **void aes\_beincrement\_x128(\_\_m128i\* counter).**

* Use overlapping keywords in the API for similar functionality, ex:

**qsc\_ntru\_generate(…)**, or **qsc\_hkdf\_generate(…)**.

* Global constants should be declared in the standard C language form of all uppercase letters, using words separated by underscores, ex. **QSC\_NTRU\_PRIVATEKEY\_SIZE**
* A constant should have no fewer than 2 words, and no more than 5 words
* Function local constants should be 6 letters, one word in uppercase ex. **CTRLEN**
* Public file names should strive to be a single word ex. **sha2**, helper files can use 2 words.
* Every public file, function, structure, enumeration, definition and type, must be documented

**Rules**

* No macros; macros are forbidden
* Excepting memory functions and callbacks, the type **void\*** shall not be used
* Use standard integer types throughout (stdint)
* Pass only the defined integer types to a function
* Use the **size\_t** integer type for every length and size parameter
* No casting to a different base integer or structure type
* Limit define-trees as much as is possible, and primarily to file includes
* Constants declared with correct integer-type and made static whenever possible
* Integrate memory functions using **SIMD**, as alternatives to *malloc*, *memcpy*, and *memset*
* Any function that can leak secret information must be constant time
* Enforce all **MISRA** rules, strive for no exceptions
* Documentation includes an overview and brief description, with all operations explained, a working example and reference links
* Document every public function, structure, enumeration, type, and constant
* Use extended function documentation flags used for auto-documentation
* Testing must be thorough and regimented, and every cryptographic primitive must be tested
* Create a separate test library that employs a testing framework, applied to all primitives and functions in the library
* Use random sampling tests, all relevant KAT tests, stress, authentication, and wellness tests for everything, including supporting tool functions