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

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

4: Tables

Contents **Page**

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 6.2a 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 6.3 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

6.5 PMAC

Header:

pmac.h

Description:

The Post-quantum Message Authentication Code generator (PMAC), is a keyed hash function, used to authenticate a message. It uses the SHA3 SHAKE as the pseudo-random function, and a Galois Field GF(2^256) multiplication and reduction functions to create a post-quantum implementation of the GMAC function.

Structures:

The **qsc_pmac_state** structure contains the internal function state.

Data Name	Data Type	Bit Length	Function
F	Uint64 Array	256	Finalization key
H	Uint64 Array	256	Hash Key
Y	Uint64 Array	256	Intermediate State
nonce	Uint8 Pointer	Variable	Nonce Pointer
noncelen	Uint64 Integer	64	Nonce Length
initialized	Boolean	8	Initialization status

Table 6.5a PMAC state structure

The **qsc_pmac_keyparams** structure contains the PMAC modes enumerals.

Data Name	Data Type	Bit Length	Function
key	Uint8 Array	256	The secret key
keylen	Uint64 Integer	64	The key length
nonce	Uint8 Array Pointer	64	The nonce pointer
noncelen	Uint64 Integer	64	The nonce length
info	Uint8 Array Pointer	64	The info pointer
infolen	Uint64 Integer	64	The info length
mode	qsc_pmac_modes	8	The mode enumeral

Table 6.5b PMAC keyparams structure

The **qsc_pmac_modes** enumeration.

Enumeration	Purpose
qsc_pmac_mode_256	The 256-bit mode
qsc_pmac_mode_512	The 512-bit mode

Table 6.5c PMAC modes

Call Hierarchy

The long form of the PMAC 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, message length, and keyparams as parameters.

```
void qsc_pmac_compute(uint8_t* output, const uint8_t* message, size_t msglen, const qsc_pmac_keyparams* keyparams)
```

Dispose

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

```
void qsc_pmac_dispose(qsc_pmac_state* ctx)
```

Finalize

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

```
void qsc_pmac_finalize(qsc_pmac_state* ctx, uint8_t* output)
```

Initialize

The initialize function initializes the state and adds the key. Takes pointers to the PMAC state, the keyparams structure as parameters.

```
void qsc_pmac_initialize(qsc_pmac_state* ctx, qsc_pmac_keyparams* keyparams)
```

Update

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

```
void qsc_pmac_update(qsc_pmac_state* ctx, const uint8_t* message, size_t msglen)
```

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 FIPS 203 specification of 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 FIPS 204 specification of the asymmetric signature-scheme 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 FIPS 205 specdification of the 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 11.2b 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 12.1 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 12.3 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 13.2a 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

Table 13.2b the collection constants

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 14.1 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 14.2a 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 14.2b font style enumeration

Constants:

Constant Name	Value	Purpose
QSC_CONSOLE_MAX_LINE	128	The maximum length of a console line.

Table 14.2c 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:

Base64, BER, DER, HEX, and PEM encoding utilities.

Enumerations:

The **qsc_encoding_ber_asn1_tag_t** enumeration

Enumeration	Purpose
BER_ASN1_BOOLEAN	BOOLEAN
BER_ASN1_EOC	End-of-Contents (EOC) marker
BER_ASN1_INTEGER	INTEGER
BER_ASN1_BIT_STRING	BIT STRING
BER_ASN1_OCTET_STRING	OCTET STRING
BER_ASN1_NULL	NULL
BER_ASN1_OBJECT_IDENTIFIER	OBJECT IDENTIFIER
BER_ASN1_OBJECT_DESCRIPTOR	Object Descriptor
BER_ASN1_EXTERNAL	EXTERNAL (or Instance-of)
BER_ASN1_REAL	REAL (floating-point)
BER_ASN1_ENUMERATED	ENUMERATED
BER_ASN1_EMBEDDED_PDV	Embedded PDV
BER_ASN1_UTF8_STRING	UTF8String
BER_ASN1_RELATIVE_OID	Relative Object Identifier
BER_ASN1_SEQUENCE	SEQUENCE and SEQUENCE OF
BER_ASN1_SET	SET and SET OF
BER_ASN1_NUMERIC_STRING	NumericString
BER_ASN1_PRINTABLE_STRING	PrintableString
BER_ASN1_T61_STRING	TeletexString (T61String)
BER_ASN1_VIDEOTEX_STRING	VideotexString
BER_ASN1_IA5_STRING	IA5String
BER_ASN1_UTCTIME	UTCTime
BER_ASN1_GENERALIZEDTIME	GeneralizedTime
BER_ASN1_GRAPHIC_STRING	GraphicString
BER_ASN1_VISIBLE_STRING	VisibleString (ISO646String)
BER_ASN1_GENERAL_STRING	GeneralString
BER_ASN1_UNIVERSAL_STRING	UniversalString
BER_ASN1_CHARACTER_STRING	CharacterString
BER_ASN1_BMP_STRING	BMPString

Table 14.3a BER ASN1 tag enumeration

Structures:

The **qsc_encoding_ber_element** structure

Data Name	Data Type	Bit Length	Function
tagclass	uint8_t	8	Tag class identifier
constructed	bool	8	Element is constructed
tagnumber	uint32_t	32	Tag number
indefinite	bool	8	Indefinate form
length	size_t	64	definite-length encoding
value	uint8_t*	8	element's raw value
children	qsc_encoding_ber_element**	variable	child pointers
ccount	size_t	64	number of child elements

Table 14.3b the BER element state structure

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)
```

BER Decode Element

Decodes a BER element from encoded data.

```
qsc_encoding_ber_element* qsc_encoding_ber_decode_element(const uint8_t* buffer,
size_t buflen, size_t* consumed)
```

BER Decode Length

Decodes a BER-encoded length value.

```
size_t qsc_encoding_ber_decode_length(const uint8_t* buffer, size_t buflen, size_t*
length, bool* indef)
```

BER Decode Tag

Decodes an ASN.1 tag from BER-encoded data.

```
size_t qsc_encoding_ber_decode_tag(const uint8_t* buffer, size_t buflen, uint8_t*
tagclass, bool* construct, uint32_t* tagnum)
```

BER Encode Element

Encodes a complete BER element.

```
size_t qsc_encoding_ber_encode_element(qsc_encoding_ber_element* element, uint8_t*
buffer, size_t buflen)
```

BER Encode Length

Encodes a length value into BER format.

```
 size_t qsc_encoding_ber_encode_length(size_t length, uint8_t* buffer, size_t
buflen)
```

BER Encode Tag

Encodes an ASN.1 tag into BER format.

```
size_t qsc_encoding_ber_encode_tag(uint8_t tagclass, bool construct, uint32_t tagnum, uint8_t* buffer, size_t buflen)
```

BER Free Element

Free a BER element from the array.

```
void encoding_ber_free_element(qsc_encoding_ber_element* element)
```

DER Decode Element

Decodes an ASN.1 element encoded in DER format.

```
qsc_encoding_ber_element* qsc_encoding_der_decode_element(const uint8_t* buffer, size_t buflen, size_t* consumed)
```

DER Encode Element

Encodes an ASN.1 element using DER (Distinguished Encoding Rules).

```
size_t qsc_encoding_der_encode_element(qsc_encoding_ber_element* element, uint8_t* buffer, size_t buflen)
```

HEX Decode

Decodes a hexadecimal string into binary data.

```
bool qsc_encoding_hex_decode(const char* input, size_t inplen, uint8_t* output, size_t otplen, size_t* declen)
```

HEX Encode

Encodes binary data into a hexadecimal string.

```
bool qsc_encoding_hex_encode(const uint8_t* input, size_t inplen, char* output, size_t otplen)
```

PEM Decode

Decodes a PEM-formatted string into binary data.

```
bool qsc_encoding_pem_decode(const char* input, uint8_t* output, size_t otplen,
size_t* declen)
```

PEM Encode

Encodes binary data in PEM format.

```
bool qsc_encoding_pem_encode(const char* label, char* output, size_t otplen, const
uint8_t* data, size_t datalen)
```

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 14.5 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 14.7 string return 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 14.8a 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 14.8b 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 14.1c 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.

Table 14.9 timestamp maximum 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 14.12c 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 15.1 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:

FlushCacheLine

Flush a cache line.

```
void qsc_memutils_flush_cache_line(void *address);
```

PrefetchL1

Pre-fetch memory to L1 cache.

```
void qsc_memutils_prefetch_l1(uint8_t* address, size_t length);
```

PrefetchL2

Pre-fetch memory to L2 cache.

```
void qsc_memutils_prefetch_l2(uint8_t* address, size_t length);
```

PrefetchL3

Pre-fetch memory to L3 cache.

```
void qsc_memutils_prefetch_l3(uint8_t* address, size_t length);
```

Malloc

Allocate a block of memory.

```
void* qsc_memutils_malloc(size_t length);
```

Realloc

Resize a block of memory.

```
void* qsc_memutils_realloc(void* block, size_t length);
```

AllocFree

Free a memory block created with alloc.

```
void qsc_memutils_alloc_free(void* block);
```

AlignedAlloc

Allocate an aligned 8-bit integer array.

```
void* qsc_memutils_aligned_alloc(int32_t align, size_t length);
```

AlignedRealloc

Reallocate an aligned 8-bit integer array.

```
void* qsc_memutils_aligned_realloc(void* block, size_t length);
```

AlignedFree

Free an aligned memory block.

```
void qsc_memutils_aligned_free(void* block);
```

Clear

Erase a block of memory.

```
void qsc_memutils_clear(void* output, size_t length);
```

ArrayUniform

Check if all array members are the same.

```
bool qsc_memutils_array_uniform(const uint8_t* input, size_t length);
```

AreEqual

Compare two byte arrays for equality.

```
bool qsc_memutils_are_equal(const uint8_t* a, const uint8_t* b, size_t length);
```

AreEqual128

Compare two 16 byte arrays for equality.

```
bool qsc_memutils_are_equal_128(const uint8_t* a, const uint8_t* b);
```

AreEqual256

Compare two 32 byte arrays for equality.

```
bool qsc_memutils_are_equal_256(const uint8_t* a, const uint8_t* b);
```

AreEqual512

Compare two 64 byte arrays for equality.

```
bool qsc_memutils_are_equal_512(const uint8_t* a, const uint8_t* b);
```

Copy

Copy a block of memory.

```
void qsc_memutils_copy(void* output, const void* input, size_t length);
```

CMul64x128

Emulates the _mm_clmulepi64_si128 intrinsic.

```
void qsc_memutils_clmulepi64_si128(uint64_t r[2], const uint64_t a[2], const
uint64_t b[2], int imm8);
```

CMul64x256AVX

Multiply two 256-bit field elements (each represented as two 128-bit integers).

```
void qsc_memutils_clmulepi64_si256_avx(__m128i r[4], const __m128i a[2], const
__m128i b[2]);
```

CMul64x256

Multiply two 256-bit field elements (each represented as two 128-bit integers).

```
qsc_memutils_clmulepi64_si256(uint64_t r[8], const uint64_t a[4], const uint64_t
b[4]);
```

GreaterThanBE128

Compare two 16 byte arrays as 128-bit big endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_be128(const uint8_t* a, const uint8_t* b);
```

GreaterThanBE256

Compare two 32 byte arrays as 256-bit big endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_be256(const uint8_t* a, const uint8_t* b);
```

GreaterThanBE512

Compare two 64 byte arrays as 512-bit big endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_be512(const uint8_t* a, const uint8_t* b);
```

GreaterThanLE128

Compare two 16 byte arrays as 128-bit little endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_le128(const uint8_t* a, const uint8_t* b);
```

GreaterThanLE256

Compare two 32 byte arrays as 256-bit little endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_le256(const uint8_t* a, const uint8_t* b);
```

GreaterThanLE512

Compare two 64 byte arrays as 512-bit little endian integers as A is greater than B.

```
bool qsc_memutils_greater_than_le512(const uint8_t* a, const uint8_t* b);
```

Move

Move a block of memory, erasing the previous location.

```
void qsc_memutils_move(void* output, const void* input, size_t length);
```

SecureErase

Erase a memory block securely.

```
void qsc_memutils_secure_erase(void* block, size_t length);
```

SecureFree

Free a secure memory block.

```
void qsc_memutils_secure_free(void* block, size_t length);
```

SecureMalloc

Allocate an secure 8-bit integer array.

```
void* qsc_memutils_secure_malloc(size_t length);
```

SetValue

Set a block of memory to a value.

```
void qsc_memutils_set_value(void* output, size_t length, uint8_t value);
```

XOR

Bitwise XOR two blocks of memory.

```
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.

```
void qsc_memutils_xorv(uint8_t* output, const uint8_t value, size_t length);
```

Zeroed

Tests an array for all zeroed elements.

```
bool qsc_memutils_zeroed(const void* input, 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