Quantum Origin C SDK - v6.0.1

Chapter 1

Quantum Origin C SDK

1.1 Quantum Origin

Quantinuum's Quantum Origin and the C SDK maximizes the strength of encryption protecting connected devices and the data they hold against advanced cyber threats. Quantum Origin is the world's only cryptographic key enhancement solution that enables you to generate quantum-computing-hardened cryptographic keys offline wherever and whenever they are needed.

The Quantum Origin C SDK allows application vendors to generate proven randomness directly within their applications.

1.1.1 How does Quantum Origin work

Quantinuum uses it's H-Series quantum computer to generate proven randomness that is used to seed a randomness extraction process within Quantum Origin. This seed has been characterized using a Bell test, proving the level of non-deterministic behavior generated on the quantum computer. This output is called the quantum seed.

The quantum seed is used by the Quantum Origin library to extract high quality, proven randomness, from a source on the local system. Some common examples of a local source of randomness are the RDSEED TRNG provided by current Intel and AMD CPUs, or CPU jitter that is present on all machines. Once deployed, Quantinuum's Quantum Origin software never needs a network connection to maintain its process.

Quantinuum has implemented the Dodis strong-seeded extractor as the randomness extractor. This process will accept the quantum seed and a local randomness source as inputs. The quantum seed is provided by Quantinuum while the local source of randomness provides an imperfect source that is refined into proven quantum-computing-hardened randomness.

The local source of randomness must be private, but the quantum seed may be public so long as its integrity is protected. Quantinuum digitally signs the quantum seed and the C SDK library validates this signature each time the quantum seed is used.

During the extraction process, each Dodis cycle produces a chunk of entropy. The underlying library will automatically adapt requests to use the small or large block size, as is appropriate for the request. The smaller block size is approximately 15x faster than the larger block size but produces 1/30th the entropy.

The C SDK allows usage of all features of Quantum Origin either when updating customers existing products or when developing new products.

Signature used to verify quantum seed	ECDSA - secp521r1 with SHA-512
Statistical distance from random (epsilon_protocol)	2^(-128)
Local randomness source minimum entropy	0.5

1.1.2 Supported Platforms

Architectures:

```
AMD64 / x86_64
armv7, armv8
```

Operating Systems:

```
Debian, and Debian based distros (e.g. Ubuntu)
Redhat and Redhat based distros (e.g. Alma)
Windows
```

Software Dependencies

The Quantum Origin requires the following:

Linux:

```
GNU C Library (glibc): v2.31 (and later)
GNU Compiler Collection (GCC): v9.4.0 (and later, recommended for building sample code)
```

Windows:

```
Microsoft Visual C++ 2015-2022 Redistributables
Visual Studio 2022 (and later, recommended for building sample code)
```

Chapter 2

File Index

2.1 File List

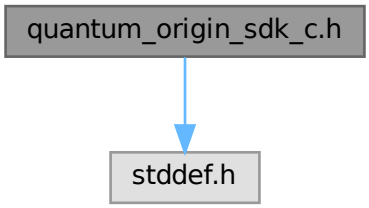
Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 quantum_origin_sdk_c.h File Reference

`#include <stddef.h>`
Include dependency graph for quantum_origin_sdk_c.h:



Macros

- #define **ERC_QOSDK_OK** 0

  Operation completed successfully.
- #define **ERC_QOSDK_UNSPECIFIED_ERROR** 19999

  An unspecified error occurred.
- #define **ERC_QOSDK_FLOOR** 13800

  Base value for error codes used by the Quantum Origin SDK C-Style API.
- #define **ERC_QOSDK_EPARAM_CONFIG_FILENAME_NOT_SUPPLIED** 13801

  A required configuration filename parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_NODE_PATH_NOT_SUPPLIED** 13802

  The required node-path parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_CONFIG_PTR_NOT_SUPPLIED** 13803

  The required configuration pointer parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_QOSDK_CONTEXT_NOT_SUPPLIED** 13804

  The required context pointer parameter was not supplied.

- #define **ERC_QOSDK_EPARAM_DEST_BUFFER_NOT_SUPPLIED** 13805

  The required destination buffer output parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_BYTES_RETURNED_PTR_NOT_SUPPLIED** 13806

  The required byte-count output parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_VALUE_PTR_NOT_SUPPLIED** 13807

  The required config value parameter was not supplied.
- #define **ERC_QOSDK_FAILED_TO_ASSIGN_SEED_SIGNATURE** 13808

  Setting the seed signature failed.
- #define **ERC_QOSDK_FAILED_TO_ASSIGN_SEED_CONTENT** 13809

  Setting the seed content failed.
- #define **ERC_QOSDK_UNSUPPORTED_OPTION** 13810

  The attempted action is not supported.
- #define **ERC_QOSDK_STD_EXCEPTION** 13811

  A standard exception occurred, see error string for more details.
- #define **ERC_QOSDK_UNKNOWN_EXCEPTION** 13812

  An unknown exception occurred.
- #define **ERC_QOSDK_EPARAM_CALLBACK_PTR_NOT_SUPPLIED** 13813

  The required callback function parameter was not supplied.
- #define **ERC_QOSDK_EPARAM_VSNPRINTF** 13814

  An error occurred whilst using vsnprintf.
- #define **ERC_QOSDK_EPARAM_MSG_BUFFER_NOT_SUPPLIED** 13815

  The required message buffer was not supplied.
- #define **ERC_QOSDK_FAILED_TO_ASSIGN_LICENSE_CONTENT** 13816

  Setting the license content failed.
- #define **ERC_QOSDK_EPARAM_INVALID_SIZE** 13817

  Size parameter is not valid.
- #define **ERC_QOSDK_EPARAM_INVALID_CACHETYPE** 13818

  Invalid or unsupported Cache Type.
- #define **ERC_QOSDK_EPARAM_HEALTH_TESTS_NOT_SUPPORTED** 13819

  Health tests are not supported in this version.
- #define **ERC_QOSDK_EPARAM_INVALID_LOGLEVEL** 13820

  Invalid or unsupported Log Level.
- #define **ERC_QOSDK_EPARAM_INVALID_LOGMODE** 13821

  Invalid or unsupported Log Mode.
- #define **ERC_QOSDK_EPARAM_INVALID_WSRTYPE** 13822

  Invalid or unsupported WSR Type.
- #define **ERC_QOSDK_EPARAM_INVALID_OPERATIONMODE** 13823

  Invalid or unsupported Operational Mode.
- #define **ERC_QOSDK_FAILED_TO_INITIALISE_LOGGING** 13824

  Failed to initialise logging subsystem.
- #define **ERC_QOSDK_PLATFORM_NOT_SUPPORTED** 13825

  Platform not supported.
- #define **ERC_QOSDK_FAILED_TO_SET_LOGGING_CALLBACK** 13826

  Failed to initialise logging subsystem.
- #define **ERC_QOSDK_ADAPTIVE_PROPORTION_HEALTH_TEST_FAILURE** 13827

  Adaptive proportion health test on WSR randomness failed.
- #define **ERC_QOSDK_REPETITION_COUNT_HEALTH_TEST_FAILURE** 13828

  Repetition count health test on WSR randomness failed.
- #define **ERC_QOSDK_OUTPUT_ADAPTIVE_PROPORTION_HEALTH_TEST_FAILURE** 13829

  Adaptive proportion health test on output randomness failed.
- #define **ERC_QOSDK_OUTPUT_REPETITION_COUNT_HEALTH_TEST_FAILURE** 13830

  Repetition count health test on output randomness failed.
- #define **ERC_QOSDK_TIMEOUT_WAITING_FOR_RANDOMNESS** 13831

  Timed out waiting for background threads to generate randomness.

Typedefs

- typedef enum tQOSDK_WSRTYPE **tQOSDK_WSRTYPE**
- typedef tQOSDK_WSRTYPE QuantumOriginWSRType
- typedef enum tQOSDK_OPMODE **tQOSDK_OPMODE**
- typedef tQOSDK_OPMODE QuantumOriginOPMode
- typedef enum tQOSDK_CACHETYPE **tQOSDK_CACHETYPE**
- typedef tQOSDK_CACHETYPE QuantumOriginCacheType
- typedef enum tQOSDK_LOGMODE **tQOSDK_LOGMODE**
- typedef tQOSDK_LOGMODE QuantumOriginLoggingMode
- typedef enum tQOSDK_LOGLEVEL **tQOSDK_LOGLEVEL**
- typedef tQOSDK_LOGLEVEL QuantumOriginLoggingVerbosity
- typedef enum tQOSDK_OPTID **tQOSDK_OPTID**
- typedef tQOSDK_OPTID QOSDK_OPT_ID
- typedef enum tQOSDK_WSR_CB_ERRCODE **tQOSDK_WSR_CB_ERRCODE**
- typedef tQOSDK_WSR_CB_ERRCODE QuantumOriginWsrCallbackErrorCode
- typedef int tQOSDK_RESULT
- typedef void tQOSDK_CTX
- typedef struct QuantumOriginConfig tQOSDK_CONFIG
- typedef int() tQOSDK_RNG_GET_WSR_DATA_FN(void *pUserData, unsigned char *pOutput, size_t outputLen)
- typedef tQOSDK_RNG_GET_WSR_DATA_FN * **tQOSDK_RNG_GET_WSR_DATA_FN_PTR**
- typedef void() tQOSDK_LOGGER_CALLBACK_FN(const tQOSDK_LOGLEVEL loglevel, const char *szMsg, size_t msgLen)
- typedef tQOSDK_LOGGER_CALLBACK_FN * **tQOSDK_LOGGER_CALLBACK_FN_PTR**

Enumerations

- enum tQOSDK_WSRTYPE { QOSDK_WSRTYPE_RDSEED = 0 , QOSDK_WSRTYPE_FILE = 1 , QOSDK_WSRTYPE_CALLBACK_FUNCTION = 2 , QOSDK_WSRTYPE_JITTER = 3 , QOSDK_WSRTYPE_QO_JITTER = 4 }

  Enum of the supported WSR types as passed to qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_WSR_TYPE, QOSDK_WSRTYPE_RDSEED);.
- enum tQOSDK_OPMODE { QOSDK_OPMODE_NORMAL = 0 , QOSDK_OPMODE_ALLZEROS = 1 }

  Enum of the available runtime operation modes of Quantum Origin.
- enum tQOSDK_CACHETYPE { QOSDK_CACHETYPE_NONE = 0 , QOSDK_CACHETYPE_CACHING = 1 , QOSDK_CACHETYPE_SYNCCACHING = 2 , QOSDK_CACHETYPE_MULTITHREAD = 3 }

  Enum of the available mechanisms available for the internal caching of randomness.
- enum tQOSDK_LOGMODE { QOSDK_LOGMODE_STDOUT = 0 , QOSDK_LOGMODE_STDERR = 1 , QOSDK_LOGMODE_SYSLOG = 2 , QOSDK_LOGMODE_DAILYFILE = 3 , QOSDK_LOGMODE_FILE = 4 , QOSDK_LOGMODE_INHERIT = 5 }

  Enum of the supported logging modes i.e. the location or subsystem to where log entries are written.
- enum tQOSDK_LOGLEVEL { QOSDK_LOGLEVEL_NONE = 0 , QOSDK_LOGLEVEL_OFF = 0 , QOSDK_LOGLEVEL_CRITICAL = 1 , QOSDK_LOGLEVEL_CRIT = 1 , QOSDK_LOGLEVEL_ERROR = 2 , QOSDK_LOGLEVEL_ERR = 2 , QOSDK_LOGLEVEL_WARNING = 3 , QOSDK_LOGLEVEL_WARN = 3 , QOSDK_LOGLEVEL_INFO = 4 , QOSDK_LOGLEVEL_DEBUG = 5 , QOSDK_LOGLEVEL_TRACE = 6 }

  Enum of the supported logging levels, which are used to control the amount of logging detail that is output.

- enum tQOSDK_OPTID {
  QOSDK_OPT_LOGGING_FILENAME = 0 , QOSDK_OPT_LOGGING_LEVEL = 1 , QOSDK_OPT_LOGGING_MODE = 2 , QOSDK_OPT_CACHE_TYPE = 3 ,
  QOSDK_OPT_CACHE_SIZE = 4 , QOSDK_OPT_CACHE_PREFILL = 5 , QOSDK_OPT_CACHE_REFILL_AT = 6 , QOSDK_OPT_WSR_TYPE = 7 ,
  QOSDK_OPT_WSR_PATH = 8 , QOSDK_OPT_HEALTH_TESTS_OUTPUT = 9 , QOSDK_OPT_SEED_SIGNATURE = 10 , QOSDK_OPT_SEED_CONTENT = 11 ,
  QOSDK_OPT_CACHE_THREAD_COUNT = 12 , QOSDK_OPT_LICENSE_DATA = 13 , QOSDK_OPT_SEED_SIGNING_KEY = 14 , QOSDK_OPT_OPERATION_MODE = 15 ,
  QOSDK_OPT_HEALTH_TESTS_DISABLED = 16 , QOSDK_OPT_HEALTH_TESTS_MAX_RETRY_COUNT = 17 , QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY = 18 , QOSDK_OPT_HEALTH_TESTS_RETRY_MULTIPLIER = 19 ,
  QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY_MAX = 20 , QOSDK_OPT_CACHE_TIMEOUT = 21 }

  Enum of supported Option IDs which are used to programmatically configure Quantum Origin SDK.
- enum tQOSDK_WSR_CB_ERRCODE {
  QOSDK_WSR_CB_SUCCESS = 0 , QOSDK_WSR_CB_FLOOR = 41000 , QOSDK_WSR_CB_RESULT_ENOMEM_OUT_OF_ = 41010 , QOSDK_WSR_CB_RESULT_EPARAM_NO_BUFFER_SUPPLIED = 41020 ,
  QOSDK_WSR_CB_RESULT_EPARAM_NO_USERDATA = 41021 , QOSDK_WSR_CB_RESULT_EINVAL_INVALID_SIZE = 41022 , QOSDK_WSR_CB_RESULT_EINVAL_INVALID_FUNCTIONPTR = 41023 , QOSDK_WSR_CB_RESULT_ERANGE_ = 41024 ,
  QOSDK_WSR_CB_RESULT_ENOINIT_NOT_INITIALISED = 41030 , QOSDK_WSR_CB_RESULT_FAILED_TO_OPEN_RAN = 41040 , QOSDK_WSR_CB_RESULT_FAILED_TO_READ_RANDOM_SOURCE = 41045 , QOSDK_WSR_CB_RESULT_FAI = 41050 ,
  QOSDK_WSR_CB_RESULT_UNSPECIFIED_ERROR = 41099 }

Functions

- tQOSDK_CTX * qosdk_init_from_config_file (const char *szConfigFilename, const char *szNodePath)
- tQOSDK_CTX * qosdk_init_from_config_struct (const tQOSDK_CONFIG *pQuantumOriginConfig)
- tQOSDK_CONFIG * qosdk_setopt_init (void)
- tQOSDK_RESULT qosdk_setopt_bytes (tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option, unsigned char *pValue, size_t cbValue)
- tQOSDK_RESULT qosdk_setopt_str (tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option, const char *szValue)
- tQOSDK_RESULT qosdk_setopt_int (tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option, size_t value)
- tQOSDK_RESULT qosdk_setopt_float (tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option, float value)
- tQOSDK_RESULT qosdk_setopt_wsr_callback (tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_↩ RNG_GET_WSR_DATA_FN_PTR pWSRCallbackFn, void *pUserData)
- void qosdk_setopt_cleanup (tQOSDK_CONFIG *pQuantumOriginConfig)
- tQOSDK_RESULT qosdk_get_randomness (tQOSDK_CTX *pQuantumOrigin, unsigned char *pBuffer, size_t cbBuffer, size_t *pBytesReturned)
- tQOSDK_RESULT qosdk_initialise_logging (tQOSDK_LOGMODE logMode, tQOSDK_LOGLEVEL logLevel, const char *szLogPath)
- tQOSDK_RESULT qosdk_set_logging_callback (tQOSDK_LOGGER_CALLBACK_FN *pLoggingCallback↩ Fn)
- void qosdk_log_message (const tQOSDK_LOGLEVEL logLevel, const char *pMsg,...)
- void qosdk_clear_logging_callback (void)
- const char * qosdk_get_error_description (void)
- tQOSDK_RESULT qosdk_get_error_code (void)
- void qosdk_destroy (tQOSDK_CTX *pQuantumOrigin)

3.1.1 Detailed Description

Quantum Origin SDK C-Style API

Quantum Origin is an entropy-enhancement engine, combining a Quantinuum-generated quantum seed with an end user's weak source of randomness to create near-perfect randomness. Quantum Origin SDK forms the core of multiple Quantinuum and Quantum Origin products, as well as being available as a standalone, offline library with a C++ or C interface.

This header file defines the API of this C-Interface.

3.1.2 Typedef Documentation

3.1.2.1 QOSDK_OPT_ID

`QOSDK_OPT_ID`

See also

tQOSDK_OPTID

3.1.2.2 QuantumOriginCacheType

`QuantumOriginCacheType`

See also

tQOSDK_CACHETYPE

3.1.2.3 QuantumOriginLoggingMode

`QuantumOriginLoggingMode`

See also

tQOSDK_LOGMODE

3.1.2.4 QuantumOriginLoggingVerbosity

`QuantumOriginLoggingVerbosity`

See also

tQOSDK_LOGLEVEL

3.1.2.5 QuantumOriginOPMode

`QuantumOriginOPMode`

See also

tQOSDK_OPMODE

3.1.2.6 QuantumOriginWsrCallbackErrorCode

`QuantumOriginWsrCallbackErrorCode`

See also

tQOSDK_WSR_CB_ERRCODE

3.1.2.7 QuantumOriginWSRType

`QuantumOriginWSRType`

See also

tQOSDK_WSRTYPE

3.1.2.8 tQOSDK_CONFIG

```
typedef struct QuantumOriginConfig tQOSDK_CONFIG
```

tQOSDK_CONFIG : A structure containing the active configuration values.

The internals of this structure are subject to change without notice and thus should not be accessed directly.

3.1.2.9 tQOSDK_CTX

```
typedef void tQOSDK_CTX
```

tQOSDK_CTX : Quantum Origin SDK object/context.

A pointer to a Quantum Origin SDK object (tQOSDK_CTX $*$) is passed to most API functions in order to identify the object context.

3.1.2.10 tQOSDK_LOGGER_CALLBACK_FN

```
typedef void() tQOSDK_LOGGER_CALLBACK_FN(const tQOSDK_LOGLEVEL loglevel, const char *szMsg,
size_t msgLen)
```

tQOSDK_LOGGER_CALLBACK_FN : A typedef of a logging callback function that will assist with the correctness of the function definition and declaration.

Notes:

1. This callback may be called from multiple threads, so should be thread safe.
2. It is recommended that a function forward declaration is added to the top of the module in which the callback is implemented so as to ensure that the interface function is correctly defined.

For example:

```
...
 // Forward declaration
static tQOSDK_LOGGER_CALLBACK_FN my_logger_callback;

// Implementation
static void my_logger_callback(const tQOSDK_LOGLEVEL loglevel, const char *szMsg, size_t msgLen)
{
    // Store or present the inbound message somewhere.
}
...
```

Parameters

loglevel	The severity of the log entry. The value is one of those in tQOSDK_LOGLEVEL.
szMsg	The null-terminated error message.
msgLen	The length of the error message (excluding the null-terminator).

Returns

void

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.2.11 tQOSDK_RESULT

```
typedef int tQOSDK_RESULT
```

tQOSDK_RESULT : Quantum Origin SDK numeric return/error code.

An enum of Quantum Origin SDK numeric error codes as returned from various API functions. For completeness, this enum includes an entry for 'success', namely ERC_QOSDK_OK, with value zero. The most recent error code can also be retrieved with a call to qosdk_get_error_code(). The qosdk_get_error_description() function can be called to help determine the nature/cause of a specific failure.

3.1.2.12 tQOSDK_RNG_GET_WSR_DATA_FN

```
typedef int() tQOSDK_RNG_GET_WSR_DATA_FN(void *pUserData, unsigned char *pOutput, size_t output↩
Len)
```

tQOSDK_RNG_GET_WSR_DATA_FN_PTR : A callback function for providing a weak source of randomness.

Notes:

1. This callback may be called from multiple threads, so should be thread safe.
2. It is recommended that a forward declaration of the form... static tQOSDK_RNG_GET_WSR_DATA_FN my↩ _wsr_callback; ...is added to the top of the module in which the callback is implemented so as to ensure that the interface function is correctly defined.

For example:

```
...
static tQOSDK_RNG_GET_WSR_DATA_FN my_wsr_callback;
static int my_wsr_callback(void *pUserData, unsigned char *pOutput, size_t outputLen)
{
    // Store outputLen bytes of randomness to the buffer pointed to by pOutput.
    return ERC_QOSDK_OK;
}
...
```

Parameters

pUserData	Pointer to arbitrary user-specified data.
pOutput	Pointer to the memory location that will receive the randomness.
outputLen	The amount of randomness that is being requested.

Returns

Must return ERC_QOSDK_OK (zero) on success, or any other value on failure.

3.1.3 Enumeration Type Documentation

3.1.3.1 tQOSDK_CACHETYPE

```
enum tQOSDK_CACHETYPE
```

Enum of the available mechanisms available for the internal caching of randomness.

Enabling the internal caching of randomness can have a significant positive effect on performance, but may have some security implications in some situations. The Quantum Origin randomness extractor produces, on demand, randomness in blocks. It is common that, after providing the amount of randomness requested, there are some bytes remaining. These could be discarded (No caching) or held in memory and be available for, or towards, a subsequent request. The potential down-side of this is that the randomness supplied in a second or subsequent call could potentially have been in memory since a previous request.

The desired caching mechanism can be set using a call to qosdk_setopt_int(), specifying the QOSDK_OPT_↩ CACHE_TYPE option and the appropriate type selected from this enum.

For example:

```
...
tQOSDK_CONFIG *pQuantumOriginConfig = NULL;
```

```
tQOSDK_RESULT erc = ERC_QOSDK_UNSPECIFIED_ERROR;

pQuantumOriginConfig = qosdk_setopt_init();
ASSERT_NE(pQuantumOriginConfig, NULL)
...
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_CACHE_TYPE, QOSDK_CACHETYPE_NONE);
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
...
pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
if (pQuantumOrigin == NULL) { Report and process error as needed }

// The config structure is no longer needed, and can be destroyed.
qosdk_setopt_cleanup(pQuantumOriginConfig);
pQuantumOriginConfig = NULL;
...
```

Enumerator

QOSDK_CACHETYPE_NONE	Disable caching of randomness.
QOSDK_CACHETYPE_CACHING	Cache randomness using a cache filled asynchronously by a background thread.
QOSDK_CACHETYPE_SYNCCACHING	Cache randomness uses a cache that is filled synchronously when a randomness request is made.
QOSDK_CACHETYPE_MULTITHREAD	Cache randomness using a cache filled asynchronously by multiple background threads.

3.1.3.2 tQOSDK_LOGLEVEL

```
enum tQOSDK_LOGLEVEL
```

Enum of the supported logging levels, which are used to control the amount of logging detail that is output.

The system outputs various log entries which are categorised into "levels". These levels range from Critical, through Error, Warning, Informative and debugging. The desired level of output can be set using a call to qosdk_setopt_int(), specifying the QOSDK_OPT_LOGGING_LEVEL option and the appropriate level selected from this enum.

See also

tQOSDK_LOGMODE

For example:

```
...
tQOSDK_CONFIG *pQuantumOriginConfig = NULL;
tQOSDK_RESULT erc = ERC_QOSDK_UNSPECIFIED_ERROR;

pQuantumOriginConfig = qosdk_setopt_init();
ASSERT_NE(pQuantumOriginConfig, NULL)
...
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_LOGGING_LEVEL, QOSDK_LOGLEVEL_WARNING);
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
...
pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
if (pQuantumOrigin == NULL) { Report and process error as needed }

// The config structure is no longer needed, and can be destroyed.
qosdk_setopt_cleanup(pQuantumOriginConfig);
pQuantumOriginConfig = NULL;
...
```

Enumerator

QOSDK_LOGLEVEL_NONE	No logging.
QOSDK_LOGLEVEL_OFF	No logging (same as QOSDK_LOGLEVEL_NONE)
QOSDK_LOGLEVEL_CRITICAL	Log only critical errors.

Enumerator

QOSDK_LOGLEVEL_CRIT	Log only critical errors (same as QOSDK_LOGLEVEL_CRITICAL)
QOSDK_LOGLEVEL_ERROR	Log all errors, including critical errors.
QOSDK_LOGLEVEL_ERR	Log all errors (same as QOSDK_LOGLEVEL_ERROR)
QOSDK_LOGLEVEL_WARNING	Log warnings, errors and critical errors.
QOSDK_LOGLEVEL_WARN	Log warnings (same as QOSDK_LOGLEVEL_WARNING)
QOSDK_LOGLEVEL_INFO	Log informative messages, warnings, errors and critical errors.
QOSDK_LOGLEVEL_DEBUG	Log debug messages, as well as all info, warning and error messages.
QOSDK_LOGLEVEL_TRACE	Log detailed trace messages, as well as all debug messages, warnings and errors.

3.1.3.3 tQOSDK_LOGMODE

```
enum tQOSDK_LOGMODE
```

Enum of the supported logging modes i.e. the location or subsystem to where log entries are written.

The system outputs various log entries as determined by the configured log level. These log entries are written to a logging "destination". The supported destinations include the console (stdout or stderr), the operating system's internal logging subsystem (syslog), a file (file or dailyfile) or to simply inherit the setting from a parent's instance of spdlog settings.

The desired mode of logging can be set using a call to qosdk_setopt_int(), specifying the QOSDK_OPT_↩ LOGGING_MODE option and the appropriate mode selected from this enum. Some modes may require additional parameters as mentioned below.

See also

tQOSDK_LOGLEVEL

For example:

```
...
tQOSDK_CONFIG *pQuantumOriginConfig = NULL;
tQOSDK_RESULT erc = ERC_QOSDK_UNSPECIFIED_ERROR;

pQuantumOriginConfig = qosdk_setopt_init();
ASSERT_NE(pQuantumOriginConfig, NULL)
...
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_LOGGING_MODE, QOSDK_LOGMODE_DAILYFILE);
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_LOGGING_FILENAME, "mylogfile.log");
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
...
pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
if (pQuantumOrigin == NULL) { Report and process error as needed }

// The config structure is no longer needed, and can be destroyed.
qosdk_setopt_cleanup(pQuantumOriginConfig);
pQuantumOriginConfig = NULL;
...
```

Enumerator

QOSDK_LOGMODE_STDOUT	Send logging to the console via stdout.
QOSDK_LOGMODE_STDERR	Send logging to the console via stderr.
QOSDK_LOGMODE_SYSLOG	Send logging to syslog.
QOSDK_LOGMODE_DAILYFILE	Send logging to a file, rotating the file each day. The name of the file is set using QOSDK_OPT_LOGGING_FILENAME.
QOSDK_LOGMODE_FILE	Send logging to a file. The name of the logfile needs to be supplied using QOSDK_OPT_LOGGING_FILENAME.
QOSDK_LOGMODE_INHERIT	Inherit logging settings from parent application (will use whichever logger is configured as spdlog default)

3.1.3.4 tQOSDK_OPMODE

```
enum tQOSDK_OPMODE
```

Enum of the available runtime operation modes of Quantum Origin.

Quantum Origin SDK can be provided in evaluation mode whereby Quantum Origin runs and generates entropy containing only NULLS. If a customer license is for evaluation mode only, then qosdk_setopt_int() must be called in the configuration setup, specifying the QOSDK_OPT_OPERATION_MODE option with the QOSDK_OPMODE_↩ ALLZEROS value.

Otherwise specifying the operational mode is optional and the mode defaults to normal run time operation i.e. Quantum Origin generates provably random quantum-derived entropy

Enumerator

QOSDK_OPMODE_NORMAL	Quantum Origin operates in normal mode.
QOSDK_OPMODE_ALLZEROS	Quantum Origin operates in evaluation mode. In this mode no genuine entropy is generated. All generated entropy will contain only NULLs.

3.1.3.5 tQOSDK_OPTID

```
enum tQOSDK_OPTID
```

Enum of supported Option IDs which are used to programmatically configure Quantum Origin SDK.

Quantum Origin SDK can be configured in various ways, including via a config file on disk, and via options specified programmatically by an application using the qosdk_setopt_xxx set of functions. The values in this enum are used to specify which of the various settings to change programmatically. Typically, an application will initialise a Quantum Origin context with a call to qosdk_setopt_init(), and then call one or more of the qosdk_setopt_[string|int|bytes]() functions with the appropriate Option ID to configure each particular setting.

For example:

```
...
tQOSDK_CONFIG *pQuantumOriginConfig = NULL;
tQOSDK_RESULT erc = ERC_QOSDK_UNSPECIFIED_ERROR;

pQuantumOriginConfig = qosdk_setopt_init();
ASSERT_NE(pQuantumOriginConfig, NULL)
...
erc = qosdk_setopt_str(pQuantumOriginConfig, QOSDK_OPT_LOGGING_FILENAME, "mylogfile.log");
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_CACHE_SIZE, 2048);
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
erc = qosdk_setopt_bytes(pQuantumOriginConfig, QOSDK_OPT_SEED_SIGNATURE, seedSignature.data(),
      seedSignature.size());
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
...
pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
if (pQuantumOrigin == NULL) { Report and process error as needed }

// The config structure is no longer needed, and can be destroyed.
qosdk_setopt_cleanup(pQuantumOriginConfig);
pQuantumOriginConfig = NULL;
...
```

Enumerator

QOSDK_OPT_LOGGING_FILENAME	String. The fully qualified logging filename.
QOSDK_OPT_LOGGING_LEVEL	Numeric. One of the values in the tQOSDK_LOGLEVEL enum.

Enumerator

QOSDK_OPT_LOGGING_MODE	Numeric. One of the values in the tQOSDK_LOGMODE enum.
QOSDK_OPT_CACHE_TYPE	Numeric. One of the values in the tQOSDK_CACHETYPE enum.
QOSDK_OPT_CACHE_SIZE	Numeric. The size of the cache in bytes.
QOSDK_OPT_CACHE_PREFILL	Numeric. The "high water mark" of the cache. Filling stops when the level exceeds this value.
QOSDK_OPT_CACHE_REFILL_AT	Numeric. The "low water mark" of the cache. Filling resumes when the level falls below this value.
QOSDK_OPT_WSR_TYPE	Numeric. The type of WSR used. The value is one of the values from the tQOSDK_WSRTYPE enum.
QOSDK_OPT_WSR_PATH	String. The path to the WSR source.
QOSDK_OPT_HEALTH_TESTS_OUTPUT	Numeric (Boolean). Whether the output health tests are run.
QOSDK_OPT_SEED_SIGNATURE	Byte array containing the seed signature.
QOSDK_OPT_SEED_CONTENT	Byte Array containing the seed itself.
QOSDK_OPT_CACHE_THREAD_COUNT	Numeric. The number of threads servicing the cache.
QOSDK_OPT_LICENSE_DATA	Byte array containing the license data.
QOSDK_OPT_SEED_SIGNING_KEY	String containing the PEM-encoded seed signing key.
QOSDK_OPT_OPERATION_MODE	Set operation mode of Quantum Origin.
QOSDK_OPT_HEALTH_TESTS_DISABLED	Numeric (Boolean). Whether the health tests are disabled.
QOSDK_OPT_HEALTH_TESTS_MAX_RETRY_↩ COUNT	Numeric. The number of times to retry if health tests are failing. Leave unset to retry indefinitely.
QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY	Numeric. The initial delay between retry attempts in milliseconds.
QOSDK_OPT_HEALTH_TESTS_RETRY_↩ MULTIPLIER	Float. The multiplier for the delay each time we back off.
QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY_↩ MAX	Numeric. The maximum delay between retry attempts in milliseconds.
QOSDK_OPT_CACHE_TIMEOUT	Numeric. Number of milliseconds to wait for randomness from the cache before failing.

3.1.3.6 tQOSDK_WSR_CB_ERRCODE

```
enum tQOSDK_WSR_CB_ERRCODE
```

Enumerator

QOSDK_WSR_CB_SUCCESS	Operation completed successfully.
QOSDK_WSR_CB_FLOOR	Base value for error codes returned by a user-define WSR callback function.
QOSDK_WSR_CB_RESULT_ENOMEM_OUT_OF↩ _MEMORY	ERROR: Memory allocation for object failed.
QOSDK_WSR_CB_RESULT_EPARAM_NO_↩ BUFFER_SUPPLIED	ERROR: Ptr to destination buffer is NULL.
QOSDK_WSR_CB_RESULT_EPARAM_NO_↩ USERDATA	ERROR: Ptr to UserData is NULL.
QOSDK_WSR_CB_RESULT_EINVAL_INVALID_↩ SIZE	ERROR: Invalid size of object.

Enumerator

QOSDK_WSR_CB_RESULT_EINVAL_INVALID_↩ FUNCTIONPTR	ERROR: Function pointer is NULL.
QOSDK_WSR_CB_RESULT_ERANGE_OUT_OF↩ _RANGE	ERROR: Value is out of range.
QOSDK_WSR_CB_RESULT_ENOINIT_NOT_↩ INITIALISED	ERROR: Subsystem/device is not initialized.
QOSDK_WSR_CB_RESULT_FAILED_TO_OPEN↩ _RANDOM_SOURCE	ERROR: Failed to open randomness source.
QOSDK_WSR_CB_RESULT_FAILED_TO_READ_↩ RANDOM_SOURCE	ERROR: Failed to read from randomness source.
QOSDK_WSR_CB_RESULT_FAILED_TO_↩ RETRIEVE_WSR_MATERIAL	ERROR: Failed to retrieve WSR material.
QOSDK_WSR_CB_RESULT_UNSPECIFIED_↩ ERROR	ERROR: Unspecified error.

3.1.3.7 tQOSDK_WSRTYPE

```
enum tQOSDK_WSRTYPE
```

Enum of the supported WSR types as passed to qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_WSR↩ _TYPE, QOSDK_WSRTYPE_RDSEED);.

Quantum Origin works by combining two sources of randomness in a very particular way - sometimes called amplification. The primary source of randomness being fed into the Quantum Origin extractor is a Quantum seed. A WSR (Weak Source of Randomness) is the name given to the secondary source of randomness. The term "weak", as used here, is relative - specifically relative to the near-perfect randomness produced from a Quantum process.

The desired second source of randomness can be set using a call to qosdk_setopt_int(), specifying the QOSDK↩ _OPT_WSR_TYPE option and the appropriate type selected from this enum. Some types may require additional parameters as mentioned below.

For example:

```
...
tQOSDK_CONFIG *pQuantumOriginConfig = NULL;
tQOSDK_RESULT erc = ERC_QOSDK_UNSPECIFIED_ERROR;

pQuantumOriginConfig = qosdk_setopt_init();
ASSERT_NE(pQuantumOriginConfig, NULL)
...
erc = qosdk_setopt_int(pQuantumOriginConfig, QOSDK_OPT_WSR_TYPE, QOSDK_WSRTYPE_RDSEED);
if (erc != ERC_QOSDK_OK) { Report and process error as needed }
...
pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
if (pQuantumOrigin == NULL) { Report and process error as needed }

// The config structure is no longer needed, and can be destroyed.
qosdk_setopt_cleanup(pQuantumOriginConfig);
pQuantumOriginConfig = NULL;
...
```

Enumerator

QOSDK_WSRTYPE_RDSEED	Use weak randomness obtained from Intel's RDSEED instruction.
QOSDK_WSRTYPE_FILE	Read weak randomness from a file (which can be a device such as /dev/urandom). The filename itself needs to be supplied using a call to qosdk_setopt_str() with parameter QOSDK_OPT_LOGGING_FILENAME.

Enumerator

QOSDK_WSRTYPE_CALLBACK_FUNCTION	Obtain weak randomness by calling a callback function. A pointer to a callback function of type tQOSDK_RNG_GET_WSR_DATA_FN needs to be supplied using the qosdk_setopt_wsr_callback() function.
QOSDK_WSRTYPE_JITTER	Obtain weak randomness by calling a dynamically loaded jitterentropy library. The library filename needs to be provided via QOSDK_OPT_WSR_PATH
QOSDK_WSRTYPE_QO_JITTER	Obtain weak randomness from the internal QO Jitter module.

3.1.4 Function Documentation

3.1.4.1 qosdk_clear_logging_callback()

```
void qosdk_clear_logging_callback (
            void  )  [extern]
```

Clear the logging callback

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.2 qosdk_destroy()

```
void qosdk_destroy (
            tQOSDK_CTX * pQuantumOrigin )  [extern]
```

qosdk_destroy() : Destroy the given Quantum Origin context.

Parameters

pQuantumOrigin	Pointer to a context previously obtained from qosdk_init_from_config_file() or qosdk_init_from_config_struct()

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.3 qosdk_get_error_code()

```
tQOSDK_RESULT qosdk_get_error_code (
            void  )  [extern]
```

qosdk_get_error_code() : Obtain the most recent error code generated by Quantum Origin.

Returns

The most recent error code

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.4 qosdk_get_error_description()

```
const char * qosdk_get_error_description (
             void  )  [extern]
```

qosdk_get_error_description() : Obtain the most recent error message generated by Quantum Origin.

Returns

A pointer to a null-terminated string containing the latest error message. The caller should *not* attempt to free this string.

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.5 qosdk_get_randomness()

```
tQOSDK_RESULT qosdk_get_randomness (
             tQOSDK_CTX * pQuantumOrigin,
             unsigned char * pBuffer,
             size_t cbBuffer,
             size_t * pBytesReturned )  [extern]
```

qosdk_get_randomness() : Request randomness from the provided Quantum Origin context.

Parameters

pQuantumOrigin	A pointer to an initialized Quantum Origin context.
pBuffer	A buffer of at least buf_size to be filled with random bytes.
cbBuffer	The size of pBuffer in bytes.
pBytesReturned	The number of bytes written to pBuffer on return.

Returns

On success, returns ERC_QOSDK_OK (0) On error, returns a non-zero error code indicating the cause of the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.6 qosdk_init_from_config_file()

```
tQOSDK_CTX * qosdk_init_from_config_file (
             const char * szConfigFilename,
             const char * szNodePath )  [extern]
```

qosdk_init_from_config_file() : Initialize a QO SDK context from a config file.

Parameters

szConfigFilename	A null terminated string specifying the relative or absolute path to the configuration file.
szNodePath	A null terminated string specifying the parent yaml node under which the configuration exists e.g. "", "/" or "generator".

Returns

On success, returns a ptr to a Quantum Origin context. This is used and required for various other calls to the Quantum Origin API. On error, returns NULL. A call to qosdk_get_error_description() and qosdk_get_error_code() can be called to determine the nature/cause of the failure.

Examples

simple_use_case_configfile_c.c.

3.1.4.7 qosdk_init_from_config_struct()

```
tQOSDK_CTX * qosdk_init_from_config_struct (
             const tQOSDK_CONFIG * pQuantumOriginConfig )  [extern]
```

qosdk_init_from_config_struct() : Initialize a Quantum Origin context from a config struct.

NOTE: All calls to the required setopt functions must be made prior to calling this function. Any subsequent calls to setopt functions after a call to qosdk_init_from_config_struct() will have no effect.

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init() and modified as required with calls to any of qosdk_setopt_int(), qosdk_setopt_str(), qosdk_setopt_bytes() and qosdk_setopt_wsr_callback() functions.

Returns

On success, returns a pointer to a Quantum Origin context. On error, returns NULL. A call to qosdk_get_error_description() and qosdk_get_error_code() can be called to determine the nature/cause of the failure.

Examples

simple_use_case_setoptions_c.c.

3.1.4.8 qosdk_initialise_logging()

```
tQOSDK_RESULT qosdk_initialise_logging (
            tQOSDK_LOGMODE logMode,
            tQOSDK_LOGLEVEL logLevel,
            const char * szLogPath )  [extern]
```

qosdk_initialise_logging() : Initialise Quantum Origin logging subsystem.

The logging subsystem must be initialised before adding, for example, a logging callback function.

In the case of the setopt approach (i.e. systems that use qosdk_init_from_config_struct()), this is always the case.

In the case of the config-file approach (i.e. systems that use qosdk_init_from_config_file()), this function needs to be called only if a logging callback, or similar, needs to be registered before a subsequent call to qosdk_init_from_config_file(). This is because the qosdk_init_from_config_file() function automatically initialises the logging subsystem during the processing of the logging entries in the config file.

Parameters

logMode	A valid logMode to use as the initial setting. This can be changed later with a call to qosdk_setopt_int(). See tQOSDK_LOGMODE above for the details of the available log modes.
logLevel	A valid logLevel to use as the initial setting. This can be changed later with a call to qosdk_setopt_int(). See tQOSDK_LOGLEVEL above for the details of the available log levels.
logPath	The path to a logfile to use as the initial setting. This parameter is used only for FILE related log modes. This can be changed later with a call to qosdk_setopt_int(). See tQOSDK_LOGLEVEL above for the details of the available log levels.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.9 qosdk_log_message()

```
void qosdk_log_message (
            const tQOSDK_LOGLEVEL logLevel,
            const char * pMsg,
             ...  )  [extern]
```

Use the Quantum Origin logger to log message.

Parameters

loglevel	The severity of the log entry. The value is one of those in tQOSDK_LOGLEVEL
psMsg	A pointer to a null-terminated string containing the log message.
...	Optional parameters to be passed into the string formatter.

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.10 qosdk_set_logging_callback()

```
tQOSDK_RESULT qosdk_set_logging_callback (
            tQOSDK_LOGGER_CALLBACK_FN * pLoggingCallbackFn )  [extern]
```

qosdk_set_logging_callback() : Register a logging callback handler.

Parameters

pLoggingCallbackFn	A pointer to a callback function of type tQOSDK_LOGGER_CALLBACK_FN, which can supply the secondary source of randomness. See tQOSDK_LOGGER_CALLBACK_FN above for the details of this callback function.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Examples

simple_use_case_configfile_c.c, and simple_use_case_setoptions_c.c.

3.1.4.11 qosdk_setopt_bytes()

```
tQOSDK_RESULT qosdk_setopt_bytes (
            tQOSDK_CONFIG * pQuantumOriginConfig,
            tQOSDK_OPTID option,
            unsigned char * pValue,
            size_t cbValue )  [extern]
```

qosdk_setopt_bytes() : Set the value of a config option whose type is bytes_t (std::vector<std::byte>).

This does a deep copy of the data pointed to by pValue to the underlying std:vector using the assign method. Consequently, the data at pValue is not required after this call, and may be destroyed or reused.

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context
option	An QOSDK_OPT_ID enum for a config option whose type is byte.
pValue	A pointer to the value to which the specified option is set.
cbValue	The number of bytes of the value.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Examples

simple_use_case_setoptions_c.c.

3.1.4.12 qosdk_setopt_cleanup()

```
void qosdk_setopt_cleanup (
            tQOSDK_CONFIG * pQuantumOriginConfig )  [extern]
```

qosdk_setopt_cleanup() : Clean up any memory allocated dynamically during the easy setopt process

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context

Examples

simple_use_case_setoptions_c.c.

3.1.4.13 qosdk_setopt_float()

```
tQOSDK_RESULT qosdk_setopt_float (
            tQOSDK_CONFIG * pQuantumOriginConfig,
            tQOSDK_OPTID option,
            float value )  [extern]
```

qosdk_setopt_float() : Set the value of a config option whose type is a float.

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context
option	An QOSDK_OPT_ID enum for a config option whose type is float.
value	The value to which the specified option is set.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

3.1.4.14 qosdk_setopt_init()

```
tQOSDK_CONFIG * qosdk_setopt_init (
            void  )  [extern]
```

qosdk_setopt_init() : Initialize a Quantum Origin config to be used in the Quantum Origin "setopt" set-up calls.

Returns

On success, a pointer an internal config context is returned. On error, returns NULL. The only cause for a potential failure is an out-of-memory condition.

Examples

simple_use_case_setoptions_c.c.

3.1.4.15 qosdk_setopt_int()

```
tQOSDK_RESULT qosdk_setopt_int (
            tQOSDK_CONFIG * pQuantumOriginConfig,
            tQOSDK_OPTID option,
            size_t value )  [extern]
```

qosdk_setopt_int() : Set the value of a config option whose type is an int.

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context
option	An QOSDK_OPT_ID enum for a config option whose type is int.
value	The value to which the specified option is set.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Examples

simple_use_case_setoptions_c.c.

3.1.4.16 qosdk_setopt_str()

```
tQOSDK_RESULT qosdk_setopt_str (
            tQOSDK_CONFIG * pQuantumOriginConfig,
            tQOSDK_OPTID option,
            const char * szValue )  [extern]
```

qosdk_setopt_str() : Set the value of a config option whose type is a string (char∗).

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context
option	An QOSDK_OPT_ID enum for a config option whose type is string.
szValue	A pointer to a null terminated string value to which the specified option is set.

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

3.1.4.17 qosdk_setopt_wsr_callback()

```
tQOSDK_RESULT qosdk_setopt_wsr_callback (
            tQOSDK_CONFIG * pQuantumOriginConfig,
            tQOSDK_RNG_GET_WSR_DATA_FN_PTR pWSRCallbackFn,
            void * pUserData )  [extern]
```

qosdk_setopt_wsr_callback() : Set the value of the WSR callback config option

Parameters

pQuantumOriginConfig	A pointer to a Quantum Origin config struct as returned from a call to qosdk_setopt_init(), containing values required to initialise a Quantum Origin context
pWSRCallbackFn	A pointer to the callback function which can supply the secondary source of randomness.
pUserData	Point to arbitrary user-specified data (will be passed on to the callback function)

Returns

On success, returns ERC_QOSDK_OK. On error, returns an error code identifying the problem. The functions qosdk_get_error_description() and qosdk_get_error_code() can be called to help determine the nature/cause of the failure.

Note that calling qosdk_setopt_wsr_callback() to set a callback function will also override any previous WSR-Type setting and force it to callback/QOSDK_WSRTYPE_CALLBACK_FUNCTION/WSRCALLBACK_FUNCTION.

3.2 quantum_origin_sdk_c.h

Go to the documentation of this file.

```
00001
00013 #pragma once
00014
00015 #include <stddef.h>
00016
00017 #ifndef ERC_QOSDK_OK
00018 #define ERC_QOSDK_OK 0
00019 #endif
00020 #ifndef ERC_QOSDK_UNSPECIFIED_ERROR
00021 #define ERC_QOSDK_UNSPECIFIED_ERROR 19999
```

```
00022 #endif
00023 #define ERC_QOSDK_FLOOR                                  13800
00024 #define ERC_QOSDK_EPARAM_CONFIG_FILENAME_NOT_SUPPLIED    13801
00025 #define ERC_QOSDK_EPARAM_NODE_PATH_NOT_SUPPLIED          13802
00026 #define ERC_QOSDK_EPARAM_CONFIG_PTR_NOT_SUPPLIED         13803
00027 #define ERC_QOSDK_EPARAM_QOSDK_CONTEXT_NOT_SUPPLIED      13804
00028 #define ERC_QOSDK_EPARAM_DEST_BUFFER_NOT_SUPPLIED        13805
00029 #define ERC_QOSDK_EPARAM_BYTES_RETURNED_PTR_NOT_SUPPLIED 13806
00030 #define ERC_QOSDK_EPARAM_VALUE_PTR_NOT_SUPPLIED          13807
00031 #define ERC_QOSDK_FAILED_TO_ASSIGN_SEED_SIGNATURE        13808
00032 #define ERC_QOSDK_FAILED_TO_ASSIGN_SEED_CONTENT          13809
00033 #define ERC_QOSDK_UNSUPPORTED_OPTION                     13810
00034 #define ERC_QOSDK_STD_EXCEPTION                          13811
00035 #define ERC_QOSDK_UNKNOWN_EXCEPTION                      13812
00036 #define ERC_QOSDK_EPARAM_CALLBACK_PTR_NOT_SUPPLIED       13813
00037 #define ERC_QOSDK_EPARAM_VSNPRINTF                       13814
00038 #define ERC_QOSDK_EPARAM_MSG_BUFFER_NOT_SUPPLIED         13815
00039 #define ERC_QOSDK_FAILED_TO_ASSIGN_LICENSE_CONTENT       13816
00040 #define ERC_QOSDK_EPARAM_INVALID_SIZE                    13817
00041 #define ERC_QOSDK_EPARAM_INVALID_CACHETYPE               13818
00042 #define ERC_QOSDK_EPARAM_HEALTH_TESTS_NOT_SUPPORTED      13819
00043 #define ERC_QOSDK_EPARAM_INVALID_LOGLEVEL                13820
00044 #define ERC_QOSDK_EPARAM_INVALID_LOGMODE                 13821
00045 #define ERC_QOSDK_EPARAM_INVALID_WSRTYPE                 13822
00046 #define ERC_QOSDK_EPARAM_INVALID_OPERATIONMODE           13823
00047 #define ERC_QOSDK_FAILED_TO_INITIALISE_LOGGING           13824
00048 #define ERC_QOSDK_PLATFORM_NOT_SUPPORTED                 13825
00049 #define ERC_QOSDK_FAILED_TO_SET_LOGGING_CALLBACK         13826
00050
00051 #define ERC_QOSDK_ADAPTIVE_PROPORTION_HEALTH_TEST_FAILURE        13827
00052 #define ERC_QOSDK_REPETITION_COUNT_HEALTH_TEST_FAILURE           13828
00053 #define ERC_QOSDK_OUTPUT_ADAPTIVE_PROPORTION_HEALTH_TEST_FAILURE 13829
00054 #define ERC_QOSDK_OUTPUT_REPETITION_COUNT_HEALTH_TEST_FAILURE    13830
00055 #define ERC_QOSDK_TIMEOUT_WAITING_FOR_RANDOMNESS                 13831
00056
00057 #ifdef __cplusplus
00058 extern "C"
00059 {
00060 #endif
00061
00063     // Note that all enum entries here are given explicit values so that, even if
00064     // one gets deprecated or removed, then the external, customer-facing API
00065     // remains unchanged.
00067
00101     typedef enum tQOSDK_WSRTYPE
00102     {
00103         QOSDK_WSRTYPE_RDSEED = 0,
00104         QOSDK_WSRTYPE_FILE   = 1,
00106         QOSDK_WSRTYPE_CALLBACK_FUNCTION = 2,
00108         QOSDK_WSRTYPE_JITTER = 3,
00110         QOSDK_WSRTYPE_QO_JITTER = 4,
00111     } tQOSDK_WSRTYPE;
00112
00117     typedef tQOSDK_WSRTYPE QuantumOriginWSRType;
00118
00130     typedef enum tQOSDK_OPMODE
00131     {
00132         QOSDK_OPMODE_NORMAL   = 0,
00133         QOSDK_OPMODE_ALLZEROS = 1,
00134     } tQOSDK_OPMODE;
00135
00140     typedef tQOSDK_OPMODE QuantumOriginOPMode;
00141
00178     typedef enum tQOSDK_CACHETYPE
00179     {
00180         QOSDK_CACHETYPE_NONE        = 0,
00181         QOSDK_CACHETYPE_CACHING     = 1,
00182         QOSDK_CACHETYPE_SYNCCACHING = 2,
00183         QOSDK_CACHETYPE_MULTITHREAD = 3,
00184     } tQOSDK_CACHETYPE;
00185
00190     typedef tQOSDK_CACHETYPE QuantumOriginCacheType;
00191
00231     typedef enum tQOSDK_LOGMODE
00232     {
00233         QOSDK_LOGMODE_STDOUT    = 0,
00234         QOSDK_LOGMODE_STDERR    = 1,
00235         QOSDK_LOGMODE_SYSLOG    = 2,
00236         QOSDK_LOGMODE_DAILYFILE = 3,
00237         QOSDK_LOGMODE_FILE      = 4,
00238         QOSDK_LOGMODE_INHERIT   = 5,
00239 #ifdef _WIN32
00240         QOSDK_LOGMODE_WINEVTLOG = 6,
00241 #endif
00242     } tQOSDK_LOGMODE;
00247     typedef tQOSDK_LOGMODE QuantumOriginLoggingMode;
00248
```

```
00281     typedef enum tQOSDK_LOGLEVEL
00282     {
00283         QOSDK_LOGLEVEL_NONE     = 0,
00284         QOSDK_LOGLEVEL_OFF      = 0,
00285         QOSDK_LOGLEVEL_CRITICAL = 1,
00286         QOSDK_LOGLEVEL_CRIT     = 1,
00287         QOSDK_LOGLEVEL_ERROR    = 2,
00288         QOSDK_LOGLEVEL_ERR      = 2,
00289         QOSDK_LOGLEVEL_WARNING  = 3,
00290         QOSDK_LOGLEVEL_WARN     = 3,
00291         QOSDK_LOGLEVEL_INFO     = 4,
00292         QOSDK_LOGLEVEL_DEBUG    = 5,
00293         QOSDK_LOGLEVEL_TRACE    = 6,
00294     } tQOSDK_LOGLEVEL;
00295
00300     typedef tQOSDK_LOGLEVEL QuantumOriginLoggingVerbosity;
00301
00338     typedef enum tQOSDK_OPTID
00339     {
00340         QOSDK_OPT_LOGGING_FILENAME              = 0,
00341         QOSDK_OPT_LOGGING_LEVEL                 = 1,
00342         QOSDK_OPT_LOGGING_MODE                  = 2,
00343         QOSDK_OPT_CACHE_TYPE                    = 3,
00344         QOSDK_OPT_CACHE_SIZE                    = 4,
00345         QOSDK_OPT_CACHE_PREFILL                 = 5,
00346         QOSDK_OPT_CACHE_REFILL_AT               = 6,
00347         QOSDK_OPT_WSR_TYPE                      = 7,
00348         QOSDK_OPT_WSR_PATH                      = 8,
00349         QOSDK_OPT_HEALTH_TESTS_OUTPUT           = 9,
00350         QOSDK_OPT_SEED_SIGNATURE                = 10,
00351         QOSDK_OPT_SEED_CONTENT                  = 11,
00352         QOSDK_OPT_CACHE_THREAD_COUNT            = 12,
00353         QOSDK_OPT_LICENSE_DATA                  = 13,
00354         QOSDK_OPT_SEED_SIGNING_KEY              = 14,
00355         QOSDK_OPT_OPERATION_MODE                = 15,
00356         QOSDK_OPT_HEALTH_TESTS_DISABLED         = 16,
00357         QOSDK_OPT_HEALTH_TESTS_MAX_RETRY_COUNT  = 17,
00358         QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY      = 18,
00359         QOSDK_OPT_HEALTH_TESTS_RETRY_MULTIPLIER = 19,
00360         QOSDK_OPT_HEALTH_TESTS_RETRY_DELAY_MAX  = 20,
00361         QOSDK_OPT_CACHE_TIMEOUT                 = 21,
00362     } tQOSDK_OPTID;
00363
00368     typedef tQOSDK_OPTID QOSDK_OPT_ID;
00369
00370     typedef enum tQOSDK_WSR_CB_ERRCODE
00371     {
00372         QOSDK_WSR_CB_SUCCESS                                = 0,
00373         QOSDK_WSR_CB_FLOOR                                  = 41000,
00374         QOSDK_WSR_CB_RESULT_ENOMEM_OUT_OF_MEMORY            = 41010,
00375         QOSDK_WSR_CB_RESULT_EPARAM_NO_BUFFER_SUPPLIED       = 41020,
00376         QOSDK_WSR_CB_RESULT_EPARAM_NO_USERDATA              = 41021,
00377         QOSDK_WSR_CB_RESULT_EINVAL_INVALID_SIZE             = 41022,
00378         QOSDK_WSR_CB_RESULT_EINVAL_INVALID_FUNCTIONPTR      = 41023,
00379         QOSDK_WSR_CB_RESULT_ERANGE_OUT_OF_RANGE             = 41024,
00380         QOSDK_WSR_CB_RESULT_ENOINIT_NOT_INITIALISED         = 41030,
00381         QOSDK_WSR_CB_RESULT_FAILED_TO_OPEN_RANDOM_SOURCE    = 41040,
00382         QOSDK_WSR_CB_RESULT_FAILED_TO_READ_RANDOM_SOURCE    = 41045,
00383         QOSDK_WSR_CB_RESULT_FAILED_TO_RETRIEVE_WSR_MATERIAL = 41050,
00384         QOSDK_WSR_CB_RESULT_UNSPECIFIED_ERROR               = 41099
00385     } tQOSDK_WSR_CB_ERRCODE;
00386
00391     typedef tQOSDK_WSR_CB_ERRCODE QuantumOriginWsrCallbackErrorCode;
00392
00401     typedef int tQOSDK_RESULT;
00402
00409     typedef void tQOSDK_CTX;
00410
00416     typedef struct QuantumOriginConfig tQOSDK_CONFIG;
00417
00446     typedef int(tQOSDK_RNG_GET_WSR_DATA_FN)(void *pUserData, unsigned char *pOutput, size_t
      outputLen);
00447     typedef tQOSDK_RNG_GET_WSR_DATA_FN *tQOSDK_RNG_GET_WSR_DATA_FN_PTR;
00448
00479     typedef void(tQOSDK_LOGGER_CALLBACK_FN)(const tQOSDK_LOGLEVEL loglevel, const char *szMsg, size_t
      msgLen);
00480     typedef tQOSDK_LOGGER_CALLBACK_FN *tQOSDK_LOGGER_CALLBACK_FN_PTR;
00481
00491     extern tQOSDK_CTX *qosdk_init_from_config_file(const char *szConfigFilename, const char
      *szNodePath);
00492
00505     extern tQOSDK_CTX *qosdk_init_from_config_struct(const tQOSDK_CONFIG *pQuantumOriginConfig);
00506
00513     extern tQOSDK_CONFIG *qosdk_setopt_init(void);
00514
00531     extern tQOSDK_RESULT qosdk_setopt_bytes(tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option,
      unsigned char *pValue, size_t cbValue);
```

```
00532
00545     extern tQOSDK_RESULT qosdk_setopt_str(tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option,
      const char *szValue);
00546
00559     extern tQOSDK_RESULT qosdk_setopt_int(tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option,
      size_t value);
00560
00573     extern tQOSDK_RESULT qosdk_setopt_float(tQOSDK_CONFIG *pQuantumOriginConfig, tQOSDK_OPTID option,
      float value);
00574
00590     extern tQOSDK_RESULT qosdk_setopt_wsr_callback(tQOSDK_CONFIG *pQuantumOriginConfig,
      tQOSDK_RNG_GET_WSR_DATA_FN_PTR pWSRCallbackFn, void *pUserData);
00591
00598     extern void qosdk_setopt_cleanup(tQOSDK_CONFIG *pQuantumOriginConfig);
00599
00612     extern tQOSDK_RESULT qosdk_get_randomness(tQOSDK_CTX *pQuantumOrigin, unsigned char *pBuffer,
      size_t cbBuffer, size_t *pBytesReturned);
00613
00637     extern tQOSDK_RESULT qosdk_initialise_logging(tQOSDK_LOGMODE logMode, tQOSDK_LOGLEVEL logLevel,
      const char *szLogPath);
00638
00649     extern tQOSDK_RESULT qosdk_set_logging_callback(tQOSDK_LOGGER_CALLBACK_FN *pLoggingCallbackFn);
00650
00658     extern void qosdk_log_message(const tQOSDK_LOGLEVEL logLevel, const char *pMsg, ...);
00659
00663     extern void qosdk_clear_logging_callback(void);
00664
00670     extern const char *qosdk_get_error_description(void);
00671
00677     extern tQOSDK_RESULT qosdk_get_error_code(void);
00678
00684     extern void qosdk_destroy(tQOSDK_CTX *pQuantumOrigin);
00685
00686 #ifdef __cplusplus
00687 }
00688 #endif
```

Chapter 4

Examples

4.1 simple_use_case_setoptions_c.c

```c
// clang-format off

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
#include <ctype.h>

#include <quantum_origin/quantum_origin_sdk_c.h>

#include "simple_use_case_setoptions_license.h"

#define BUFFER_SIZE 400U

// Forward declaration to ensure that the interface function is correctly defined.
static tQOSDK_LOGGER_CALLBACK_FN my_qo_logger_callback;


//-------------------------------------------------------------
// Logger callback
//-------------------------------------------------------------
static void my_qo_logger_callback(const tQOSDK_LOGLEVEL logLevel, const char *szMsg, size_t msgLen)
{
    // Store or present the inbound message somewhere.
    switch(logLevel)
    {
        case QOSDK_LOGLEVEL_NONE     : break;
        case QOSDK_LOGLEVEL_CRITICAL : fprintf(stderr, "[QuantumOrigin] CRITICAL: %.*s\n", (int)msgLen,
      szMsg); break;
        case QOSDK_LOGLEVEL_ERROR    : fprintf(stderr, "[QuantumOrigin] ERROR: %.*s\n", (int)msgLen, szMsg);
      break;
        case QOSDK_LOGLEVEL_WARNING  : fprintf(stderr, "[QuantumOrigin] WARNING: %.*s\n", (int)msgLen,
      szMsg); break;
        case QOSDK_LOGLEVEL_INFO     : fprintf(stderr, "[QuantumOrigin] INFO: %.*s\n", (int)msgLen, szMsg);
      break;
        case QOSDK_LOGLEVEL_DEBUG    : fprintf(stderr, "[QuantumOrigin] DEBUG: %.*s\n", (int)msgLen, szMsg);
      break;
        default:
        case QOSDK_LOGLEVEL_TRACE    : fprintf(stderr, "[QuantumOrigin] TRACE: %.*s\n", (int)msgLen, szMsg);
      break;
    }
}

//-------------------------------------------------------------
// Simple hex dump
//-------------------------------------------------------------
static void hex_dump(const char *szTitle, const uint8_t *pData, size_t cbData)
{
    printf("%s [%lu bytes]: ", szTitle ? szTitle : "HEXDUMP: ", (unsigned long)cbData);
    if (pData && cbData)
    {
        for (int ii = 0; ii < cbData; ii++)
        {
            printf("%2.2X ", pData[ii]);
        }
```

```
    }
    printf("\n");
}

//------------------------------------------------------------
// Main
//------------------------------------------------------------
int main(int argc, char **argv)
{
    tQOSDK_CTX *    pQuantumOrigin       = NULL;
    uint32_t        bytesRequested       = 0xFFFFFFFF;
    size_t          bytesSupplied        = 0;
    unsigned char   resultBuffer[BUFFER_SIZE];
    int             erc                  = ERC_QOSDK_OK;
    tQOSDK_CONFIG *pQuantumOriginConfig = NULL;

    // Parse parameters
    for (int jj = 1; jj < argc; jj++)
    {
        if (strcmp(argv[jj], "--help") == 0)
        {
            printf("USAGE: simple_use_case_setoptions_c [<BytesOfRandomness>] // (Default = 32 bytes)\n");
            return EXIT_SUCCESS;
        }
        if (bytesRequested == 0xFFFFFFFF)
        {
            if (argv[jj][0] == '-')
            {
                fprintf(stderr, "ERROR: Number of bytes requested cannot be negative\n");
                return EXIT_FAILURE;
            }
            bytesRequested = (uint32_t)atol(argv[jj]);
            if (bytesRequested > BUFFER_SIZE)
            {
                fprintf(stderr, "ERROR: Number of bytes requested must be <= %u\n", BUFFER_SIZE);
                return EXIT_FAILURE;
            }
            continue;
        }
    }

    if (bytesRequested == 0xFFFFFFFF)
    {
        bytesRequested = 32;
    }

    // Initialise Quantum Origin Logging
    erc = qosdk_initialise_logging(QOSDK_LOGMODE_STDERR, QOSDK_LOGLEVEL_WARNING, NULL);
    if (erc)
    {
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to initialize logging (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        return EXIT_FAILURE;
    }
    // Register a Quantum Origin Logging callback
    erc = qosdk_set_logging_callback(my_qo_logger_callback);
    if (erc)
    {
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to set logging callback (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        return EXIT_FAILURE;
    }

    // Initialise Quantum Origin
    pQuantumOriginConfig = qosdk_setopt_init();
    if (!pQuantumOriginConfig) { erc = ERC_QOSDK_UNSPECIFIED_ERROR; goto cleanup_and_exit; }
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_LOGGING_MODE    , QOSDK_LOGMODE_STDERR    ); if
      (erc) { goto cleanup_and_exit; } // e.g. [STDOUT, STDERR, SYSLOG, DAILYFILE, FILE, INHERIT]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_LOGGING_LEVEL   , QOSDK_LOGLEVEL_ERR      ); if
      (erc) { goto cleanup_and_exit; } // e.g. [NONE, CRITICAL, ERROR, WARNING, INFO, DEBUG, TRACE]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_CACHE_TYPE      , QOSDK_CACHETYPE_CACHING ); if
      (erc) { goto cleanup_and_exit; } // e.g. [CACHING, SYNCCACHING, MULTITHREAD, NONE]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_CACHE_SIZE      , 10240                   ); if
      (erc) { goto cleanup_and_exit; } // e.g. [1048576]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_CACHE_PREFILL   , 10240                   ); if
      (erc) { goto cleanup_and_exit; } // e.g. [128]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_CACHE_REFILL_AT , 2048                    ); if
      (erc) { goto cleanup_and_exit; } // e.g. [1024]
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_WSR_TYPE        , QOSDK_WSRTYPE_RDSEED    ); if
      (erc) { goto cleanup_and_exit; } // e.g. [QO_JITTER, RDSEED, FILE, JITTER, CALLBACK_FUNCTION]
    erc = qosdk_setopt_bytes(pQuantumOriginConfig, QOSDK_OPT_LICENSE_DATA     , (uint8_t *)licenseContent,
      sizeof(licenseContent) ); if (erc) { goto cleanup_and_exit; }
    erc = qosdk_setopt_int   (pQuantumOriginConfig, QOSDK_OPT_OPERATION_MODE  , QOSDK_OPMODE_NORMAL     ); if
      (erc) { goto cleanup_and_exit; }
    pQuantumOrigin = qosdk_init_from_config_struct(pQuantumOriginConfig);
```

```
    if (pQuantumOrigin == NULL)
    {
        erc = qosdk_get_error_code();
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to initialize Quantum Origin with qosdk_init_from_config_struct()
      (QuantumOriginErrorMsg=%s)\n", szQuantumOriginError?szQuantumOriginError:"Unspecified");
        goto cleanup_and_exit;
    }
    // The pQuantumOriginConfig structure is no longer needed, and can be destroyed.
    qosdk_setopt_cleanup(pQuantumOriginConfig);
    pQuantumOriginConfig = NULL;

    // Exercise the logging callback
    qosdk_log_message(QOSDK_LOGLEVEL_WARNING, "This is a test warning message");

    // Get randomness
    bytesSupplied = 0;
    erc = qosdk_get_randomness(pQuantumOrigin, resultBuffer, bytesRequested, &bytesSupplied);
    if (erc)
    {
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to get randomness. (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        goto cleanup_and_exit;
    }

    // Print results
    hex_dump("RESULT", resultBuffer, bytesSupplied);

    erc = ERC_QOSDK_OK;

cleanup_and_exit:
    // Cleanup and exit
    if (pQuantumOriginConfig)
    {
        qosdk_setopt_cleanup(pQuantumOriginConfig);
        pQuantumOriginConfig = NULL;
    }
    if (pQuantumOrigin)
    {
        qosdk_destroy(pQuantumOrigin);
        pQuantumOrigin = NULL;
    }
    qosdk_clear_logging_callback();

    if (erc != ERC_QOSDK_OK)
    {
        fprintf(stderr, "ERROR: simple_use_case_setoptions_c failed with errCode=%d\n", erc);
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

4.2 simple_use_case_configfile_c.c

```
// clang-format off

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
#include <ctype.h>

#include <quantum_origin/quantum_origin_sdk_c.h>

#define BUFFER_SIZE 400U

// Forward declaration to ensure that the interface function is correctly defined.
static tQOSDK_LOGGER_CALLBACK_FN my_qo_logger_callback;


//-------------------------------------------------------------
// Logger callback
//-------------------------------------------------------------
static void my_qo_logger_callback(const tQOSDK_LOGLEVEL logLevel, const char *szMsg, size_t msgLen)
{
    // Store or present the inbound message somewhere.
    switch(logLevel)
    {
        case QOSDK_LOGLEVEL_NONE     : break;
```

```
        case QOSDK_LOGLEVEL_CRITICAL : fprintf(stderr, "[QuantumOrigin] CRITICAL: %.*s\n", (int)msgLen,
      szMsg); break;
        case QOSDK_LOGLEVEL_ERROR    : fprintf(stderr, "[QuantumOrigin] ERROR: %.*s\n", (int)msgLen, szMsg);
      break;
        case QOSDK_LOGLEVEL_WARNING  : fprintf(stderr, "[QuantumOrigin] WARNING: %.*s\n", (int)msgLen,
      szMsg); break;
        case QOSDK_LOGLEVEL_INFO     : fprintf(stderr, "[QuantumOrigin] INFO: %.*s\n", (int)msgLen, szMsg);
      break;
        case QOSDK_LOGLEVEL_DEBUG    : fprintf(stderr, "[QuantumOrigin] DEBUG: %.*s\n", (int)msgLen, szMsg);
      break;
        default:
        case QOSDK_LOGLEVEL_TRACE    : fprintf(stderr, "[QuantumOrigin] TRACE: %.*s\n", (int)msgLen, szMsg);
      break;
    }
}

//------------------------------------------------------------------
// Simple hex dump
//------------------------------------------------------------------
static void hex_dump(const char *szTitle, const uint8_t *pData, size_t cbData)
{
    printf("%s [%lu bytes]: ", szTitle ? szTitle : "HEXDUMP: ", (unsigned long)cbData);
    if (pData && cbData)
    {
        for (int ii = 0; ii < cbData; ii++)
        {
            printf("%2.2X ", pData[ii]);
        }
    }
    printf("\n");
}

//------------------------------------------------------------------
// Main
//------------------------------------------------------------------
int main(int argc, char **argv)
{
    tQOSDK_CTX *    pQuantumOrigin       = NULL;
    uint32_t        bytesRequested       = 0xFFFFFFFF;
    size_t          bytesSupplied        = 0;
    unsigned char   resultBuffer[BUFFER_SIZE];
    int             erc                  = ERC_QOSDK_UNSPECIFIED_ERROR;
    char *          szConfigFilename     = "config-sample-01.yml";

    // Parse parameters
    for (int jj = 1; jj < argc; jj++)
    {
        if (strcmp(argv[jj], "--help") == 0)
        {
            printf("USAGE: simple_use_case_configfile_c [<BytesOfRandomness>] // (Default = 32 bytes)\n");
            return EXIT_SUCCESS;
        }
        if (bytesRequested == 0xFFFFFFFF)
        {
            if (argv[jj][0] == '-')
            {
                fprintf(stderr, "ERROR: Number of bytes requested cannot be negative\n");
                return EXIT_FAILURE;
            }
            bytesRequested = (uint32_t)atol(argv[jj]);
            if (bytesRequested > BUFFER_SIZE)
            {
                fprintf(stderr, "ERROR: Number of bytes requested must be <= %u\n", BUFFER_SIZE);
                return EXIT_FAILURE;
            }
            continue;
        }
    }

    if (bytesRequested == 0xFFFFFFFF)
    {
        bytesRequested = 32;
    }

    // Initialise Quantum Origin Logging
    erc = qosdk_initialise_logging(QOSDK_LOGMODE_STDERR, QOSDK_LOGLEVEL_WARNING, NULL);
    if (erc)
    {
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to initialize logging (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        return EXIT_FAILURE;
    }
    // Register a Quantum Origin Logging callback
    erc = qosdk_set_logging_callback(my_qo_logger_callback);
    if (erc)
    {
```

```
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to set logging callback (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        return EXIT_FAILURE;
    }

    // Initialise Quantum Origin
    pQuantumOrigin = qosdk_init_from_config_file(szConfigFilename, "");
    if (pQuantumOrigin == NULL)
    {
        erc = qosdk_get_error_code();
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to initialize Quantum Origin with qosdk_init_from_config_file(\"%s\")
      (QuantumOriginError=%d: \"%s\")\n", szConfigFilename, erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        // Cleanup and exit
        qosdk_clear_logging_callback();
        return EXIT_FAILURE;
    }


    // Exercise the logging callback
    qosdk_log_message(QOSDK_LOGLEVEL_WARNING, "This is a test warning message");

    // Get randomness
    bytesSupplied = 0;
    erc = qosdk_get_randomness(pQuantumOrigin, resultBuffer, bytesRequested, &bytesSupplied);
    if (erc)
    {
        const char *szQuantumOriginError = qosdk_get_error_description();
        fprintf(stderr, "ERROR: Failed to get randomness. (QuantumOriginError=%d: \"%s\")\n", erc,
      szQuantumOriginError?szQuantumOriginError:"Unspecified");
        // Cleanup and exit
        qosdk_destroy(pQuantumOrigin);
        qosdk_clear_logging_callback();
        return EXIT_FAILURE;
    }

    // Print results
    hex_dump("RESULT", resultBuffer, bytesSupplied);

    // Cleanup and exit
    qosdk_destroy(pQuantumOrigin);
    qosdk_clear_logging_callback();

    return EXIT_SUCCESS;
}
```

Index