Document QMS3570

QOS 2.2 libcxx Certified C++ Library

Safety Manual

Version 7.0

Document Number	QMS3570
Version	7.0
Project ID	LIBCXX_ASILD
Document Generated	2022-06-29
Document Status	Approved

2022-06-29

Table of Contents

CHAPTER 1

Introduction

Contents

1.1 This Document

1.1.1 Document Identification

This document is the Safety Manual for the BlackBerry QNX implementation of a library of C++ functions specified in ISO 14882:—[1]. This document also uses the abbreviated name, "*libcxx*", to refer to this BlackBerry product. It is identified by the unique number QMS3570 together with the version number 7.0.

1.1.2 Document Purpose

This safety manual describes specific constraints on the use of *libcxx* that must be followed for the top-level safety claim made by BlackBerry QNX for this project to remain valid.

This document is not a "programming guide" for *libcxx*. Nothing in this document should be regarded as information about the functional behavior of any *libcxx* function. Similarly, this document does not describe any performance, capacity or other limitations of this

product. The absence of references to such limitations must not be interpreted to imply that there are no such limitations. This product relies exclusively on ISO 14882:— as the authoritative source of information about the functional behavior of *libcxx* functions and its limitations.

The use of a qualified software component such as this product provides a degree of confidence about the avoidance of defects in the implementation of this software. However, the use of *libcxx* will not make any application "inherently safe". For example, an application that uses this library might include hazard causes unrelated to the functionality of *libcxx*. Similarly, an application might use elements of this product incorrectly or inappropriately, or make assumptions about its functionality that are not supported by the contents of ISO 14882:—.

A list of headers which restricts the qualification of this product to those *libcxx* functions whose signature appears in one of the headers identified can be found in revision list contained in the latest QOS 2.2 appendix accessible at

https://fs-products.tuvasi.com/certificates?cert_id=6694.

The system integrator and application developer must read and understand the contents of this document and the constraints it places on the use of this product.

1.2 Document Audience

The intended audience of this manual is:

1. The system integrator responsible for the integration of the C++ Library into a product (hereafter referred to as "the system integrator").
2. The individual or group within the product organization responsible for assessing the safety of the product.

Before using the BlackBerry QNX implementation of a C++ Library (*libcxx*) within a certified system or system to be certified, technical personnel SHALL:

1. Ensure that *libcxx* has been installed and is used in accordance with the associated installation guide.
2. Understand how to use ISO 14882:— as a source of information about the functionality of *libcxx* functions.

1.3 Applicability

This document applies to *libcxx* which is the BlackBerry QNX implementation of a C++ Library.

1.4 Nomenclature

The key words “MUST", “MUST NOT", “REQUIRED", “SHALL", “SHALL NOT", “SHOULD", “SHOULD NOT", “RECOMMENDED", “MAY", and “OPTIONAL" used within chapters 3 and 4 are to be interpreted as described in RFC 2119 [2] and clarified in RFC 8174 [3], both published by the Internet Engineering Task Force (IETF) and available at http://www.rfc-editor.org/rfc/.

1.5 Scope

This document refers to the latest release of the *QOS 2.2 libcxx Certified C++ Library*.

1.6 Glossary

- ISO 14882:— notation refers to C++ Standards including ISO/IEC 14882: 2011, 2014 and 2017
- ISO 26262 notation refers to ISO 26262 standard
- BlackBerry stands for BlackBerry Limited
- QOS stands for QNX OS for Safety

CHAPTER 2

libcxx Safety & Product Scope

This chapter defines the environment within which it is assumed that the *libcxx* will be installed.

It also presents:

1. the product scope for *libcxx*
2. safety goals for the *libcxx*

Additional information on the Installation and runtime environment is provided in the *libcxx* User's Guide.

2.1 Product Scope

libcxx is a software component adapted from the *LLVM* open source implementation of the *C++ Standard Library* that was developed outside BlackBerry QNX and without the use of BlackBerry QNX software development processes. However, it has been maintained by BlackBerry QNX.

In addition to the main *libcxx.so* library, there are several other third party libraries on which it depends:

- *libcatalog*: This library is used in *<locale>*.
- *libm*: C implementation of mathematics functions.
- *libc*: C library
- *libsupc++*: Support library for *libcxx* distributed as part of the *GCC* compiler. This library implements clause 18 of ISO 14882:—, the *Itanium C++ ABI*, and additional QNX-specific customizations.

- *crt* files*: Low level runtime routines used by the compiler. There are C++ specific runtime files.

Figure 2.1 shows the various components and their relationships. Only the components within the "C++ Component Scope" box are in scope for this analysis i.e.:

- libcxx headers files (i.e., C++/v1 header files in green box) are ASIL-B
- libcxx runtime binaries (i.e., libc++.so, libc++.a in red box) are ASIL-D
- libsupc++ library accessed by libcxx (i.e., libsupc++.a in red box) is ASIL-D.

The additional components shown are either out of certification scope or previously certified.

Note that the BlackBerry *QNX Software Development Platform* also provides non-certified implementations of the in-scope libraries and headers.

The *libcxx* Hazard and Risk Analysis is exclusively concerned with the following three top-level safety claims:

- The available test evidence from BlackBerry QNX implementation of the C++ library, *libcxx*, is demonstrably adequate to support a conclusion that functions provided by *libcxx* library are correct according to ISO 14882:—.
- Binaries produced using the *q++* compiler and certified header files behave according to the ISO 14882:— standard.
- Failure modes for implementation-specific handling of memory exhaustion during exception processing in *libsupc++* are considered and adequately addressed.

2.2 Safety Goals

The *libcxx*, when used in accordance with the constraints given in chapter 3 of this document within the environment described in this chapter, has been designed to meet the requirements of ISO 14882:—[1] and ISO 26262 for a component to be used in items in order to realize safety goals up to classification ASIL-D (ISO 26262) as applicable to the *libcxx* run-rime binaries, and ASIL-B (ISO 26262) as applicable to all other elements of *libcxx*.

2.3 Assumed Environment

The *libcxx* is intended for use to build applications for the *QNX OS for Safety* (QOS), on systems that are compliant with the *QOS Safety Manual* and any other associated manuals published by BlackBerry QNX.

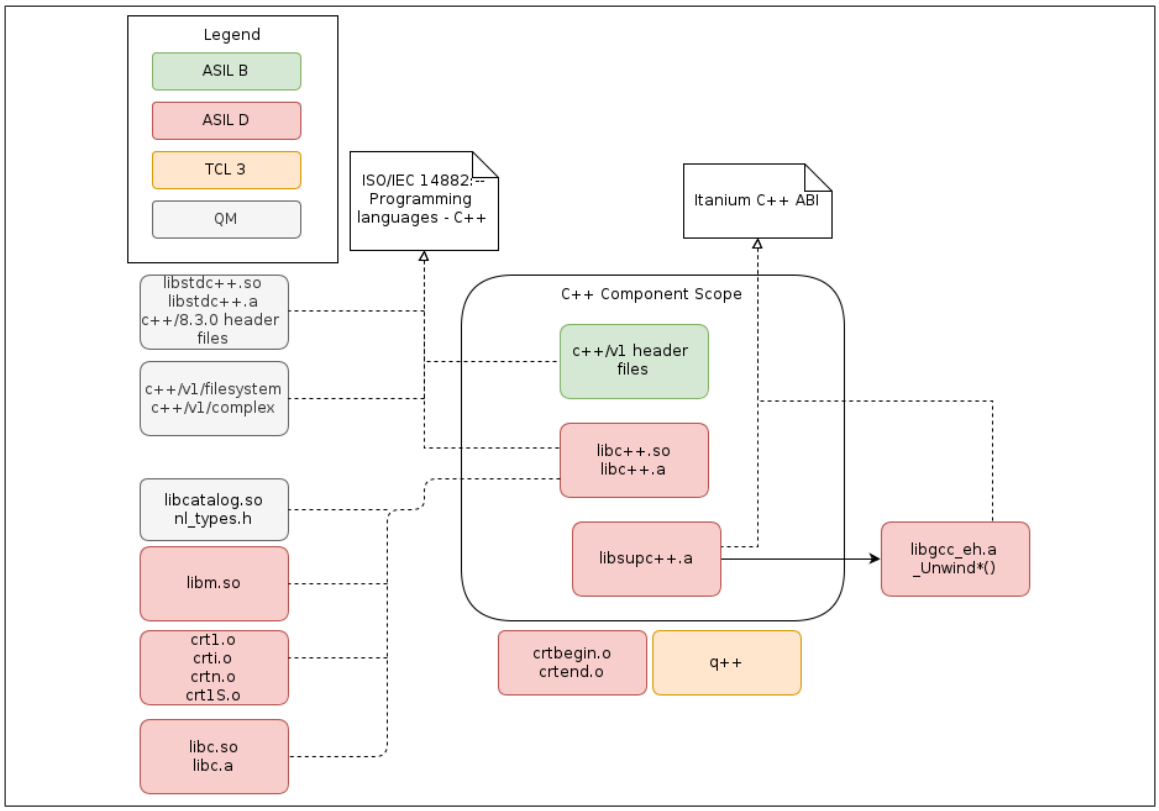

Figure 2.1: Product Scope

2.4 Installation

The "as shipped" *libcxx* product includes release notes, a safety manual (this document), C++ source code contained in *libcxx* header files, object code contained in *libcxx* static library (`libc++.a`) and shared object files (`libc++.so`). Refer to the "Installing and Using QOS" guide at

`http://www.qnx.com/download/feature.html?programid=49542`.

CHAPTER 3

Restrictions

Contents

3.1 Introduction

The system integrator shall read and understand the contents of this Safety Manual and the constraints it places on all uses of this product.

3.2 General Restrictions

Restriction 1 The system integrator SHALL know the content of release notes shipped by BlackBerry QNX for the particular release being used, that includes known deviations from ISO 14882:—[1].

If the system integrator is uncertain about the implications of a release note on the design or implementation of a system subject to the requirements of ISO 14882:—[1], then they should approach BlackBerry QNX for an explanation.

3.3 Installation and Use Restrictions

Restriction 2 The system integrator SHALL ensure the *libcxx* dependencies defined in the scope are the versions shipped with the *libcxx* library.

The libcxx library depends on a few other libraries, namely:

- libcatalog: This library is used in <locale>.
- libm: C implementation of mathematics functions
- libc: C library
- libsupc++: Support library for libcxx distributed as part of the GCC compiler. This library implements clause 18 of ISO 14882:—, the Itanium C++ ABI, and additional BlackBerry QNX specific customization.
- crt* files: Low level runtime routines used by the compiler. There are C++ specific runtime files.

For predictable and known behavior, it's important to ensure that the versions of above supporting libraries being used are same that were provided with the *libcxx* library, failure to do so will invalidate the safety configuration.

Note that libcatalog is not in scope of certification (see Restrictions 14 and 15).

Restriction 14 The `-lcatalog` compiler option SHALL NOT be used in a certified application.

Restriction 15 `do_get`, `do_open`, and `do_close` functions SHALL NOT be used for accessing message catalog via `std::messages`.

Restriction 16 libcxx functions SHALL NOT be called from interrupt handlers or signal handlers.

3.4 Architectural and Design Restrictions

None.

3.5 Implementation Restrictions

Restriction 3 The application developer SHALL NOT use internal functions and classes when compiling and linking applications with *libcxx*. The internal functions and classes are identified with the following naming conventions:

- names that begin with a double underscore
- names that begin with an underscore followed by an upper-case letter
- names that begin with an underscore when defined in the global namespace

Some of the internal functions might be visible to the application developer while their behavior is not defined in ISO 14882:—. As such, it is necessary that an application developer check that the application source code does not reference such classes/functions. As the qualification of this product only applies to functions specified in ISO 14882:—, it is necessary for the system integrator to confirm that only functions specified in ISO 14882:— are used from this product.

Restriction 4 The system integrator SHALL compile the *libcxx* with one of the following compiler options: `-std=c++11`, `-std=c++14`, `-std=c++17`.

It is necessary to avoid accidental use of a wrong library or header files. For example: The flag "-std=c++17" to the compiler indicates revision C++17 of ISO/IEC 14882 standard is being used.

Restriction 5 The system integrator SHALL prevent use of `-stdlib=libstdc++` and `-Y_gpp` compiler options with *libcxx*.

As `libstdc++` and `-Y_gpp` are GNU standard C++ Library, and are not part of the certification scope for *libcxx* library, usage of these compiler options is restricted for a safety certified application.

Restriction 6 The system integrator SHALL prevent modifications to the configuration of the compiler that would result in precedence being given to a non-certified C++ Standard Library with the following compiler options:

- include paths (`-I`, being uppercase i)
- library path (`-L`)
- library name (`-l`, being lowercase L)

As the GNU Standard C++ Library (libstdc++) is not in scope of *libcxx* certification.

Restriction 7 The system integrator SHALL prevent use of the filesystem header defined in C++ 17 in a safety context.

The filesystem library is implemented in a separate library, libc++fs, and header, <filesystem>, for libcxx. This library is not in certification scope despite being part of the C++17 specification.

Restriction 8 The system integrator SHALL prevent use of the C++ <complex> and <complex.h> header for complex mathematics functions.

The standard C++ <complex> and <complex.h> header are out of certification scope. User can consider using `libm` mathematics library provided with *libcxx* library.

Restriction 9 The system integrator SHALL prevent use of `-fno-exception` and `-fno-rtti` compiler options with *libcxx*.

The above options are not in scope of *libcxx* certification.

Restriction 10 The system integrator SHALL prevent redefinition of `_LIBCXX` macros from source code or from the command line, or from both.

Redefining macros involves ensuring the new definition of macro must be effectively the same as the old macro. And, redefining macros could lead to compiler warnings and errors. Hence, redefining of `_LIBCXX` macros must not be done to maintain known behavior of *libcxx* library.

Restriction 11 The developer of application, compiling and linking with libcxx SHALL ensure only the qualified qcc or q++ compiler delivered with QNX OS for Safety is used.

Restriction 12 The default terminate handler provided with libcxx SHALL be overridden with a new implementation of the terminate handler.

The default terminate handler, `__verbose_terminate_handler`, is currently not certified. Hence, the responsibility is on the user to explicitly provide and certify their own `std::terminate_handler` function.

Restriction 13 The overridden terminate handler function SHALL be set with `std:set_terminate`.

CHAPTER 4

Recommendations

Contents

4.1 Introduction

This chapter lists recommended practices for using this product.

4.2 General Recommendations

Recommendation 1 If the system integrator observes that *libcxx* is not behaving in accordance with its published documentation, the system integrator SHOULD report this observation to BlackBerry QNX.

For any product potentially used in safety-critical applications, it is important to be aware of defects. The mechanism for reporting such defects is given in Appendix B of this document.

4.3 Installation and Use Recommendations

Recommendation 2 The system integrator SHOULD consider the use of appropriate guidelines in the safe use of the C++ programming language, as per Chapter 5, Part 6 of

ISO 26262.

The use of appropriate guidance, as recommended above, can help reduce opportunities for incorrect or inappropriate uses of *libcxx* functionality that could potentially contribute to unsafe behavior. However, compliance with such guidance does not make any application "inherently safe".

4.4 Architectural and Design Recommendations

Recommendation 3 The system integrator SHOULD perform additional unit testing and integration testing, for every user-defined type as a template argument to a *libcxx* class/function instantiation.

Especially with user-defined type there is a scope of error falling through the cracks. So, it is recommended in order to verify the correctness of the implementation of the functions that the template argument requires. Not doing additional testing for user-defined types might lead to unknown behavior.

Recommendation 4 The system integrator SHOULD account for the possibility that the processing of an exception could cause termination of the process without warning.

Exception handling could lead to memory exhaustion, and if that occurs system will call `terminate()` without warning. For more information refer Appendix A of this document.

Recommendation 6 Any complexity requirements from ISO 14882:— SHOULD be verified, if necessary, in the context of an integrated system.

The complexity requirements refer to the time and/or space complexity and ISO 14882:— provide more details of all such functions. By above recommendation, the user is being made aware that the burden of testing for such functions is on them.

4.5 Implementation Recommendations

Recommendation 5 The system integrator SHOULD prevent use of implicit type conversions as arguments to libcxx functions, and use static analysis tool to prevent such conversion, in all their applications that uses *libcxx* functions.

The errors caused due to implicit type conversion are hard to detect and costly affair so avoid them, and use a static analysis tool for more confidence.

APPENDIX A

libcxx Memory Exhaustion

Memory exhaustion can be defined as a state of the system where system runs out of allocated memory resources to process or handle any new allocations or exceptions.

When exceptions are thrown by the code, storage is needed for exception handling. This storage must persist while stack is being unwound, since it will be utilized by the handler, and must be thread-safe. Hence, the exception object storage/memory is normally allocated in the heap. While allocating memory for exception objects; under low memory conditions, memory exhaustion may occur. An example code for memory allocation to an exception object would be:

```
void *__cxa_allocate_exception(size_t thrown_size);
```

It's important to note that if memory exhaustion occurs, the system will first attempt to use pre-allocated emergency buffer allocated using the `malloc()` call made at static storage initialization time. The emergency buffer is only used under the following conditions:

- The exception object representation is under 1024 bytes
- the current thread currently holds 4 or fewer emergency buffers
- there are fewer than 16 other threads holding emergency buffers

Under any other condition, `terminate()` is called immediately.

The memory exhaustion, exception object allocation, and emergency buffer are implemented as per Itanium C++ ABI Specification 2.4.2 [Allocating the Exception Object] and 3.3.1 [Exception Storage].

APPENDIX B

Support Information

Contents

B.1 Support Information

BlackBerry QNX is committed to providing customers with exceptional support.

Support is available through a dedicated online portal, person-to-person help lines, community portal, knowledge base, and more. Here are some useful links:

Support portal — central hub for information, resources and issue management for all registered support plan customers: http://www.qnx.com/account/login.html

Support options — your choice of person-to-person help lines, dedicated technical resources, or a customized team of support professionals: http://www.qnx.com/support/support.html

Knowledge base — clearinghouse for tips, tricks, and technical articles: http://www.qnx.com/support/knowledgebase.html

B.2 New to BlackBerry QNX?

If you have not had the opportunity to avail yourself of BlackBerry QNX support services in the past, here are some helpful starting points:

Accessing Online Technical Support — your first stop for help with setting up your myQNX account, registering your support plan, and using the portal: http://www.qnx.com/download/feature.html?programid=18239

Seven Steps to Developing a QNX Program: Quickstart Guide – an indispensable resource for help with installing and configuring the QNX Software Development Platform : http://www.qnx.com/developers/docs/7.0.0/#com.qnx.doc.qnxsdp.quickstart/topic/about.html

B.3 Reporting Defects

As stated by Recommendation 4.2 on Page 12, you are recommended to report defects that you find in the *libcxx* so that BlackBerry QNX can assess the possible impact of the defect on the continued safe operation of the *libcxx*.

Such defects should be reported through your support contact, whether standard or priority.

B.4 Direct Contacts

For questions regarding this Safety Manual or *QOS 2.2 libcxx Certified C++ Library* ISO 26262 certification, contact BlackBerry QNX via your support contact or by calling +1-613-591-0931.

Bibliography

[1] ISO/IEC 14882 - Information technology – Programming languages – C++. Technical report, International Organisation for Standardization, Geneva, Switzerland, 2011, 2014, 2017.

[2] S. Bradner. RFC2119: Key words for use in RFCs to Indicate Requirement Levels, 1997.

[3] B. Leiba. RFC8174: Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words, 2017.

BlackBerry | QNX