TECHNICAL SPECIFICATION

ISO/IEC TS 19216

First edition
2018-04

Programming Languages — C++ Extensions for Networking

Langages de programmation — Extensions C++ pour mise en réseau

Reference number
ISO/IEC TS 19216:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see `www.iso.org/directives`).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see `www.iso.org/patents`).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: `www.iso.org/iso/foreword.html`.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

1 Scope [scope]

1 This document describes extensions to the C++ Standard Library. This document specifies requirements for implementations of an interface that computer programs written in the C++ programming language may use to perform operations related to networking, such as operations involving sockets, timers, buffer management, host name resolution and internet protocols. This document is applicable to information technology systems that can perform network operations, such as those with operating systems that conform to the POSIX interface. This document is applicable only to vendors who wish to provide the interface it describes.

2 Normative references [references]

1 The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

(1.1) — ISO/IEC 14882:2014, *Programming languages — C++*

(1.2) — ISO/IEC TS 19568:2015, *C++ Extensions for Library Fundamentals*

(1.3) — ISO/IEC 9945:2009, *Information Technology — Portable Operating System Interface (POSIX)*

(1.4) — ISO/IEC 2382-1:1993, *Information technology — Vocabulary*

2 The programming language and library described in ISO/IEC 14882 is herein called the C++ Standard. References to clauses within the C++ Standard are written as "C++ 2014, Clause 17". The operating system interface described in ISO/IEC 9945 is herein called POSIX.

3 This document mentions commercially available operating systems for purposes of exposition. POSIX® is a registered trademark of The IEEE. Windows® is a registered trademark of Microsoft Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of these products.

4 Unless otherwise specified, the whole of the C++ Standard's Library introduction (C++ 2014, Clause 17) is included into this document by reference.

3 Terms and definitions [defs]

1 For the purposes of this document, the terms and definitions given in ISO/IEC 14882:2014, ISO/IEC 2382-1:1993, and the following apply.

2 ISO and IEC maintain terminological databases for use in standardization at the following addresses:

(2.1) — IEC Electropedia: available at http://www.electropedia.org/

(2.2) — ISO Online browsing platform: available at http://www.iso.org/obp

3 Terms that are used only in a small portion of this document are defined where they are used and italicized where they are defined.

3.1 **[defs.host.byte.order]**

host byte order

the arrangement of bytes in any integer type when using a specific machine architecture

[SOURCE: ISO/IEC 9945:2009, 3.193]

3.2 **[defs.net.byte.order]**

network byte order

the way of representing any integer type such that, when transmitted over a network via a network endpoint, the `int` type is transmitted as an appropriate number of octets with the most significant octet first, followed by any other octets in descending order of significance

[SOURCE: ISO/IEC 9945:2009, 3.237]

3.3 **[defs.sync.op]**

synchronous operation

operation where control is not returned until the operation completes

3.4 **[defs.async.op]**

asynchronous operation

operation where control is returned immediately without waiting for the operation to complete

[*Note 1 to entry:* Multiple asynchronous operations may be executed concurrently. — *end note*]

4 General Principles [general]

4.1 Conformance [conformance]

1 Conformance is specified in terms of behavior. Ideal behavior is not always implementable, so the conformance subclauses take that into account.

4.1.1 POSIX conformance [conformance.9945]

1 Some behavior is specified by reference to POSIX. How such behavior is actually implemented is unspecified.

2 [*Note:* This constitutes an "as if" rule allowing implementations to call native operating system or other APIs. — *end note*]

3 Implementations are encouraged to provide such behavior as it is defined by POSIX. Implementations shall document any behavior that differs from the behavior defined by POSIX. Implementations that do not support exact POSIX behavior are encouraged to provide behavior as close to POSIX behavior as is reasonable given the limitations of actual operating systems and file systems. If an implementation cannot provide any reasonable behavior, the implementation shall report an error as specified in Error Reporting (9).

4 [*Note:* This allows users to rely on an exception being thrown or an error code being set when an implementation cannot provide any reasonable behavior. — *end note*]

5 Implementations are not required to provide behavior that is not supported by a particular operating system.

4.1.2 Conditionally-supported features [conformance.conditional]

1 This document defines conditionally-supported features, in the form of additional member functions on types that satisfy `Protocol` (18.2.6), `Endpoint` (18.2.4), `SettableSocketOption` (18.2.9), `GettableSocketOption` (18.2.8) or `IoControlCommand` (18.2.12) requirements.

2 [*Note:* This is so that, when the additional member functions are available, C++ programs can extend the library to add support for other protocols and socket options. — *end note*]

3 For the purposes of this document, implementations that provide all of the additional member functions are known as extensible implementations.

4 [*Note:* Implementations are encouraged to provide the additional member functions, where possible. It is intended that POSIX and Windows implementations will provide them. — *end note*]

4.2 Acknowledgments [intro.ack]

1 The design of this specification is based, in part, on the Asio library written by Christopher Kohlhoff.

5 Namespaces and headers [namespaces]

1 The components described in this document are experimental and not part of the C++ standard library. All components described in this document are declared in namespace `std::experimental::net::v1` or a sub-namespace thereof unless otherwise specified. The headers described in this document shall import the contents of `std::experimental::net::v1` into `std::experimental::net` as if by:

```
namespace std {
  namespace experimental {
    namespace net {
      inline namespace v1 {}
    }
  }
}
```

2 Unless otherwise specified, references to other entities described in this document are assumed to be qualified with `std::experimental::net::v1::`, references to entities described in the C++ standard are assumed to be qualified with `std::`, and references to entities described in C++ Extensions for Library Fundamentals are assumed to be qualified with `std::experimental::fundamentals_v2::`.

6 Future plans (Informative) [plans]

1 This clause describes tentative plans for future versions of this document and plans for moving content into future versions of the C++ Standard.

2 The C++ committee may release new versions of this document, containing networking library extensions we hope to add to a near-future version of the C++ Standard. Future versions will define their contents in `std::experimental::net::v2`, `std::experimental::net::v3`, etc., with the most recent implemented version inlined into `std::experimental::net`.

3 When an extension defined in this or a future version of this document represents enough existing practice, it will be moved into the next version of the C++ Standard by replacing the `experimental::net::v`*N* segment of its namespace with `net`, and by removing the `experimental/` prefix from its header's path.

7 Feature test macros (Informative) [feature.test]

1 These macros allow users to determine which version of this document is supported by the headers defined by the specification. All headers in this document shall define the `__cpp_lib_experimental_net` feature test macro in Table 1.

2 If an implementation supplies all of the conditionally-supported features specified in 4.1.2, all headers in this document shall additionally define the `__cpp_lib_experimental_net_extensible` feature test macro.

Table 1 — Feature-test macro(s)

Macro name	Value
`__cpp_lib_experimental_net`	201707
`__cpp_lib_experimental_net_extensible`	201707

8 Method of description (Informative) [description]

1 This subclause describes the conventions used to specify this document, in addition to those conventions specified in C++ 2014, 17.5.

8.1 Structure of each clause [structure]

8.1.1 Detailed specifications [structure.specifications]

1 In addition to the elements defined in C++ 2014, 17.5.1.4, descriptions of function semantics contain the following elements (as appropriate):

(1.1) — *Completion signature:* if the function initiates an asynchronous operation, specifies the signature of a completion handler used to receive the result of the operation.

8.2 Other conventions [conventions]

8.2.1 Nested classes [nested.class]

1 Several classes defined in this document are nested classes. For a specified nested class `A::B`, an implementation is permitted to define `A::B` as a synonym for a class with equivalent functionality to class `A::B`. [*Note:* When `A::B` is a synonym for another type `A` provides a nested type `B`, to emulate the injected class name. — *end note*]

9 Error reporting [err.report]

9.1 Synchronous operations [err.report.sync]

1 Most synchronous network library functions provide two overloads, one that throws an exception to report system errors, and another that sets an `error_code` (C++ 2014, 19.5).

[*Note:* This supports two common use cases:

(1.1) — Uses where system errors are truly exceptional and indicate a serious failure. Throwing an exception is the most appropriate response.

(1.2) — Uses where system errors are routine and do not necessarily represent failure. Returning an error code is the most appropriate response. This allows application specific error handling, including simply ignoring the error.

— *end note*]

2 Functions not having an argument of type `error_code&` report errors as follows, unless otherwise specified:

(2.1) — When a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications, the function exits via an exception of a type that would match a handler of type `system_error`.

(2.2) — Destructors throw nothing.

3 Functions having an argument of type `error_code&` report errors as follows, unless otherwise specified:

(3.1) — If a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications, the `error_code&` argument `ec` is set as appropriate for the specific error. Otherwise, the `ec` argument is set such that `!ec` is `true`.

4 Where a function is specified as two overloads, with and without an argument of type `error_code&`:

```
R f(A1 a1, A2 a2, ..., AN aN);
R f(A1 a1, A2 a2, ..., AN aN, error_code& ec);
```

5 then, when R is non-`void`, the effects of the first overload are as if:

```
error_code ec;
R r(f(a1, a2, ..., aN, ec));
if (ec) throw system_error(ec, S);
return r;
```

6 otherwise, when R is `void`, the effects of the first overload are as if:

```
error_code ec;
f(a1, a2, ..., aN, ec);
if (ec) throw system_error(ec, S);
```

7 except that the type thrown may differ as specified above. `S` is an NTBS indicating where the exception was thrown. [*Note:* A possible value for `S` is `__func__`. — *end note*]

8 For both overloads, failure to allocate storage is reported by throwing an exception as described in the C++ standard (C++ 2014, 17.6.5.12).

9 In this document, when a type requirement is specified using two function call expressions *f*, with and without an argument `ec` of type `error_code`:

