# Template for comments and secretariat observations

| Date:2017-02-17 | Document: WG21 N4643 | Project: 19216 |
|---|---|---|

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| US 001 | | | | General | The TS travels in terms of *function objects* rather than Callable. The C++ 17 standard is moving towards a more general specification. Rather than a function object, anything that can be processed by *INVOKE* with the appropriate signature should be allowed. | Use *invokable* terminology throughout the TS instead of *function object*. | |
| GB 1 002 | | | | Ge | The BSI would like to ensure that outstanding issues on the issues lists are all considered before the final TS is produced | | |
| CA1 003 | | | | GE | We do not have technical comments on the document, however, we note that the format of the document is inconsistent with the ISO IEC Directives part 2, and with the ISO Online Browsing Platform, which makes the following clauses available publically, clauses 1, 2 and three. | Restructure the document such that<br>Clause 1 is scope<br>Clause 2 is Normative References<br>Clause 3 is Terms and Definitions<br>Place other material in clauses 4, 5, 6, etc. | |
| FR1 004 | | | 13.2.2<br><br>13.2.7 | te | Implementing asynchronous operations or executors as described in paragraphs 13.2.2 and 13.2.7 could be perfomed with the use of coroutines.<br><br>It would be necessary to clarify or assess the complexity of this code if it was to use one or other of the different proposals now faced (PDTS 22277 and P009R1) | | |
| US 005 | | 03 | 2 | te | The normative reference to the POSIX standard should not be a non-normative note. | Either add another paragraph with a normative reference to the POSIX standard, or remove the [Note – end note] mark-up on this paragraph. | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| US 006 | | 13.02 | | ge | The 'CompletionToken' mechanism used to determine the return types of initiating functions has teachability drawbacks due to its complexity. Also, there would be a high cost if users were to replicate this kind of interface in medium and higher-level libraries built upon the Networking TS. A simpler interface whereby initiating functions have invokable parameters is sufficiently extendable, as they are, to work with futures. | Make initiating functions use invokable callbacks instead of completion tokens as in Boost.ASIO. | |
| GB 2 007 | | 13.02.1 | | Te | "No constructor ... shall exit via an exception" over-specification.<br>This is probably a copy/paste error introduced when this text was copied from the standard (I believe from [allocator.requirements]).<br>The intent is only that copy and move constructors shall not throw, and I think that is already covered by "copy operation, move operation". Other constructors not covered by the ProtoAllocator? requirements should be allowed to throw. | Delete "constructor," so that the sentence reads "No comparison operator, copy operation, move operation, or swap operation on these types shall exit via an exception." | |
| GB 3 008 | | 13.02.2 | | Te | Reentrancy and run/dispatch.<br>In 17.6.5.8 [reentrancy] the C++14 standard says:<br>"Except where explicitly specified in this standard, it is implementation-defined which functions in the Standard C++ library may be recursively reentered."<br>In the executor requirements, the intention is that the dispatch() function may be recursively reentered. A statement to this effect may need to be added to the requirements. (All dispatch() member functions provided by executors in the TS itself should then by implication allow reentrancy.) | Explicitly specify that dispatch functions can be recursively re-entered. | |
| GB 4 009 | | 13.02.2 | | Te | "No constructor ... shall exit via an exception" over-specification.<br>This is probably a copy/paste error introduced when this text was copied from the standard (I believe from [allocator.requirements]). | Delete "constructor," so that the sentence reads "No comparison operator, copy operation, move operation, swap operation, or member functions context, on_work_started, and on_work_finished on these types shall exit via an exception." | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | The intent is only that copy and move constructors shall not throw, and I think that is already covered by "copy operation, move operation". Other constructors not covered by the Executor requirements should be allowed to throw. | | |
| GB 5 010 | | 13.02.2 | | Te | Executor requirements table refers to undefined name 'Func'.<br>The Executor requirements table entries for 'dispatch', 'post' and 'defer' refer to an undefined name 'Func', in 'DECAY_COPY(forward<Func>(f))'. This name is not listed in the paragraph preceding the table. | Fix the requirements table so that either the name 'Func' is defined, or so that the requirements for 'dispatch', 'post' and 'defer' are specified without referring to this type. | |
| US 011 | | 13.02.2 async.reqmts.executor | Table 4 | ge | Without knowing more about what kind of executor is being used, a user will have difficulty deciding which of the three functions to use to add a task to an executor. It is preferable to limit the options for adding tasks to a generic executor. Concrete executors can add additional functions if required. | Remove the defer function from executors, as that is the least well-defined. This would match the existing Boost ASIO implementation. | |
| US 012 | | 13.02.3 async.reqmts.executioncontext | Table 5 | te | Missing ~ in row 2 | x. X() should be x.~X() | |
| US 013 | | 13.02.4 async.reqmts.service | paragraph 5 | te | "user-defined function objects" - user defined is possibly over-specification. In other parts of the TS it refers to types not specified by this TS. If there are function objects defined in the standard, they should also be destroyed. | remove "user-defined" | |
| US 014 | | 13.02.7.12 async.reqmts.async.completio | Para 2 | te | This seems to be respecifying *INVOKE* | Reword using *INVOKE* | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| **MB/ NC**[1] | **Line number** | **Clause/ Subclause** | **Paragraph/ Figure/Table** | **Type of comment**[2] | **Comments** | **Proposed change** | **Observations of the secretariat** |
|---|---|---|---|---|---|---|---|
| | | n | | | | | |
| GB 6 015 | | 13.07 | | Te | Reentrancy and use_service/make_service.<br>The intention is that both use_service and make_service may make nested calls (from the Service constructor) to use_service or make_service. Obviously these nested calls will require a different Service template argument. I am uncertain if calling these function templates with different template arguments counts as recursive reentrance, but if it does then we may need to add a sentence explicitly specifying that this is permitted. | Decide if it's needed and add a suitable sentence. | |
| GB 7 016 | | 14.02.1 | | Te | run()/run_one() specification overly restrictive on users.<br><br>Both the run() and run_one() functions include the following statement:<br><br>"Must not be called from a thread that is currently calling a run function."<br><br>This restriction was originally added as a way to prevent users from entering a kind of "deadlock". This is because run() and run_one() can block until the io_context runs out of work. Since outstanding work includes currently executing function objects, if a function object makes a nested call to run()/run_one() that nested call could block forever as the work count can never reach zero.<br><br>However, it has been brought to Chris Kohlhoff's attention by users that there are valid use cases for making these nested calls. Deadlock can be avoided if some other condition will cause | Strike those sentences from both those places. Make it the responsibility of the user to avoid the conditions for deadlock. | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | run()/run_one() to exit (e.g. an exception, explicit call to stop, run_one finished running a single function, etc). This condition can be known ahead of time by the user.<br><br>The existing implementation in asio does not make any beneficial use of this restriction. | | |
| GB 8 017 | | 14.02.1 | | Te | Reentrancy and run functions.<br>The intention is that the run functions may be recursively reentered. We may want add a sentence explicitly specifying this. | Explicitly specify that run functions can be recursively re-entered. | |
| GB 9 018 | | 16 | | Te | user-provided overloads of buffer_size intended?<br>Is it intended that users can provide overloads of buffer_size for user-defined buffer sequence types? Is it something we should consider? I'm thinking about basic_streambuf and user defined alternatives, it can compute buffer_size for its input and output sequences in constant time. | Consider making buffer_size a customization point. | |
| GB 10 019 | | 16.02 | | Te | Relax strict aliasing requirement for user-defined buffer sequence iterators.<br>See LWG issue 2779. | See LWG issue 2779. | |
| GB 11 020 | | 16.02, 16.7 | | Te | Consider adding noexcept to buffer sequence requirements<br>[buffer.reqmts.mutablebuffersequence], [buffer.reqmts.constbuffersequence], [buffer.seq.access] | Adding "Shall not exit via an exception." to the the requirements for net::buffer_sequence_begin(x) and 'net::buffer_sequence_end(x).<br>Requiring that the conversion of the iterator value type to const_buffer or mutable_buffer should not exit via exception.<br>(And perhaps place the same requirement on the iterator traversal and dereference too, although I'm not sure if this is already implied elsewhere in the standard?)<br>Adding noexcept to the buffer sequence access functions.<br>The current implementation in asio assumes that these never throw, and in general I think low level | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | | buffer operations should not throw. | |
| US 021 | | 16.02.1<br>16.2.2 | | te | buffer.requirements.mutablebuffersequence<br>buffer.requirements.constbuffersequence | Please address LEWG 2779. | |
| GB 12 022 | | 16.05 | | Te | const_buffer is a view<br>const_buffer [buffer.const] is a non-owning type that consists of a pointer and a size.<br>In the same vein, it would make sense to rename mutable_buffer [buffer.mutable] to something like buffer_span. While there is no naming precedence for this in C++17, there are various span proposals (e.g. P0122R1 and P0123R1.) | Consider renaming const_buffer to buffer_view partly to indicate that users must keep the underlying, owning buffer alive during operations that involve const_buffer, and partly for naming consistency with string_view. | |
| US 023 | | 16.05 [buffers.const ] | | ed | const_buffer operator+= is missing from the index of library names | Add const_buffer operator+= to the index of library names. | |
| GB 13 024 | | 18 | | Te | The derived socket types basic_datagram_socket and basic_stream_socket should specify that their native_handle_type is the same as basic_socket::native_handle_type. | Add such specification | |
| GB 14 025 | | 18.05 | | Te | Add integer_option helper.<br>When using socket options that are not defined by [socket.opt], users have to create their own class that follows the GettableSocketOption? [socket.reqmts.gettablesocketoption] or SettableSocketOption? [socket.reqmts.settablesocketoption] concepts.<br>As the majority of socket options are integral, it would ease the burden of using such socket options if an integer_option helper class was available (basically the same as asio::detail::socket_option::integer.) Its use would boil down to:<br>using maximum_segment_size_type = net::integer_option<IPPROTO_TCP, | | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | TCP_MAXSEG>;<br>maximum_segment_size_type mtu;<br>socket.get_option(mtu);<br>A similar helper class for boolean socket options could also be added. | | |
| GB 15 026 | | 18.06, 18.9 | | Te | Consider adding release() member functions to basic_socket and basic_socket_acceptor<br>Historically, Asio has not provided this facility as it could not be portably implemented using the preferred OS-specific mechanisms. Specifically, on Windows, once a socket was associated to an I/O completion port it could not be disassociated. This means that the socket could not be truly released for arbitrary use by the user. However, as of Windows 8.1, the ability to disassociate a socket is available (via the kernel API NtSetInformationFile?). This means that release() can be portably implemented. | Add the following member functions to both basic_socket and basic_socket_acceptor:<br>native_handle_type release();<br>native_handle_type release(error_code& ec);<br>with effects that any pending asynchronous operations are cancelled and ownership of the native handle is transferred to the caller. | |
| GB 16 027 | | 18.06, 18.9 | | Te | Consider adding constructors to basic_socket and basic_socket_acceptor to move a socket to another io_context<br>Historically, Asio has not provided this facility as it could not be portably implemented using the preferred OS-specific mechanisms. Specifically, on Windows, once a socket was associated to an I/O completion port it could not be disassociated. This means that the socket could not be moved from one io_context to another. However, as of Windows 8.1, the ability to disassociate a socket is available (via the kernel API NtSetInformationFile?). This means that the ability to move sockets between io_contexts can be portably implemented. | Add constructors to basic_socket (and to derived classes basic_stream_socket, basic_datagram_socket), and to basic_socket_acceptor, e.g.:<br>basic_socket(io_context& ctx, basic_socket&& rhs);<br>with effect that pending asynchronous operations are cancelled on rhs, and then ownership of the underlying socket is transferred to the newly created socket object with the associated io_context. | |
| GB 17 028 | | 18.09.1 | | Te | [socket.acceptor.cons] move ctor missing postcondition.<br>The postconditions for the basic_socket_acceptor move ctor don't have any postconditions on | Add "native_handle()} returns the prior value of rhs.native_handle()." | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | native_handle(). | | |
| GB 18 029 | | 19.01.1 | | Te | [socket.streambuf.cons] Add missing error() postconditions.<br>The basic_socketstreambuf constructors do not give any postconditions for the ec_ member. | Add "and !error()" to paragraphs 2 and 4. Add "error() == rhs_p.error()" and "!rhs_a.error()" to paragraph 6. | |
| GB 19 030 | | 19.01.1 | | Te | [socket.streambuf.cons] Cover changes to rhs in operator= effects.<br>The assignment operator for basic_socketstreambuf doesn't state the effects on the RHS. | Change "*this has the observable state it would have had if it had been move constructed from rhs" to "*this and rhs have the observable state they would have had if *this had been move constructed from rhs" | |
| US 031 | | 20.01 socket.algo.connect | para 1 and 2 | te | template parameter InputIterator not used in declaration, but EndpointSequence is. | change class InputIterator to class EndpointSequence in error code versions. | |
| US 032 | | 20.02 socket.algo.async.connect | Para 1 | te | The second version of async_connect also uses InputIterator as the template parameter, where it should be EndpointSequence. | change class InputIterator to class EndpointSequence | |
| GB 20 033 | | 21.01 | | Te | Shorten ip::resolver_errc enumerator names.<br>These enumerator names predate enum classes as a language feature and were so named to eliminate likely name clashes with other entities in the same namespace. The enumerator "host_not_found_try_again" is particularly long and could be shortened. | Rename the enumerator "host_not_found_try_again" to "try_again". | |
| GB 21 034 | | 21.04.3, 21.6.3 | | Te | Consider ip::address::is_loopback() and ip::address_v6::is_loopback() behaviour for IPv4-mapped IPv6 addresses<br>Currently the ip::address_v6::is_loopback() and ip::address::is_loopback() functions return false for an IPv4-mapped IPv6 address that maps IPv4 loopback. This behaviour follows the | Consider whether to alter the specification of ip::address::is_loopback() and ip::address_v6::is_loopback() such that they additionally return true if passed an IPv4 loopback address as an IPv4-mapped IPV6 address. | |

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|
| | | | | | implementation of the IN6_IS_ADDR_LOOPBACK macro and RFC 4291. An Asio user has proposed that these functions should return true for IPv4 loopback addresses that are mapped to IPv4-mapped IPv6 address. | | |
| US 035 | | 21.11 [internet.network.v4] | | ed | The argument name in operator<< is not consistent with the argument name in operator<< in [internet.network.v4.io] | Change the argument name from “addr” to “net” | |
| US 036 | | 21.12 [internet.network.v6] | | ed | The argument name in operator<< is not consistent with the argument name in operator<< in [internet.network.v6.io] | Change the argument name from “addr” to “net” | |
| US 037 | | 21.12, 21.12.04 [internet.network.v6] | | ed | There is a cut-and-paste error in in the declaration of make_network_v6. “string_v6” has been used instead of “string_view”. | Change the occurrences of “const string_v6&” to “string_view” | |
| US 038 | | 21.21.01 [internet.multicast.outbound] | | ed | There is a cut-and-paste error in the description of the name() member function. The document says “*_MULTICAST_HOPS” where “*_MULTICAST_IF” is intended. | Change “IPV6_MULTICAST_HOPS” to “IPV6_MULTICAST_IF”, and “IP_MULTICAST_HOPS” to “IP_MULtiCAST_IF” | |

D:\ISO\data\prod_iso_comment-collation\work\temp\ISO_IEC PDTS 19216_AFNOR.docx: Collation successful

D:\ISO\data\prod_iso_comment-collation\work\temp\ISO_IEC PDTS 19216_ANSI.docx: Collation successful

D:\ISO\data\prod_iso_comment-collation\work\temp\ISO_IEC PDTS 19216_BSI.doc: Collation successful

D:\ISO\data\prod_iso_comment-collation\work\temp\ISO_IEC PDTS 19216_SCC.doc: Collation successful

Collation of files was successful. Number of collated files: 4

SELECTED (number of files): 4

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

| Date:2017-02-17 | Document: WG21 N4643 | Project: 19216 |
|---|---|---|

| MB/ NC[1] | Line number | Clause/ Subclause | Paragraph/ Figure/Table | Type of comment[2] | Comments | Proposed change | Observations of the secretariat |
|---|---|---|---|---|---|---|---|

PASSED TEST (number of files): 4

FAILED TEST (number of files): 0

CCT - Version 4.0/2015

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial