Doc No. P0488R0
Date: 2016-10-19
Project: Programming Language C++
Reply To: Barry Hedquist, beh@peren.com

Subject: WG21 Working Paper, NB Comments, ISO/IEC CD 14882

Attached is a WG21 Working Paper containing National Body Comments on ISO/IEC CD 14882, Programming Language C++. These comments are identical to those submitted to ISO/IEC, except that the comments are numbered, so we know what we are talking about. In the case of the ANSI Comments, large excerpts in the “Proposed Change” column were redacted and replaced with a hot link to the proposal ‘p’ paper. That was done for readability. In another case, the Project Number was changed from something to 14882.

Thanks everyone,
Barry Hedquist
beh@peren.com

Template for comments and secretariat observations

Date: 11/10/2016	Document:	Project:ISO 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
ES 1		7.1.6	1,3	Te	The proposed feature of inline variables goes beyond the original problem to be solved. That is, avoiding the need to provide a definition for any static data member (constexpr or not) from a class.	Remove inline variables from C++17. Solve exclusively the multiple definitions of: a) Constexpr data members b) Static data members	
ES 2		8.5	1	Te	While structured bindings are a very useful feature the latest syntax after last minute modification make it more complex and less uniform. The use of bracktes may introduce problems with attributes and lambdas	Reconsider the braces syntax instead of the brackets syntax.	

ES 3		D.1	1	Ed	Example should use constexpr for variable declaration.	Change:
						
						struct A {
						static constexpr int n = 5; // definition (declaration in C++ 2014)
						};
						const int A::n; //
						
						to:
						
						struct A {
						static constexpr int n = 5; // definition (declaration in C++ 2014)
						};
						constexpr int A::n; //	

ES 4				Ge	Concepts is a highly relevant feature with field experience. We strongly support the introduction of Concepts to C++17. If such introduction is considered impossible, we suggest Concepts TS is introduced at the beginning of the process for the	Adopt Concepts TS for C++17. Alternatively consider introducing it in the draft for the next standard.	

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

Template for comments and secretariat observations

Date: 11/10/2016	Document:	Project:ISO 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					next standard.		
ES 5				Ge	Unified syntax call provides a simplification mechanism and would allow simplifications to many libraries.	Consider separately the two halves of unified syntax call	
ES 6				Ge	Operator dot provides important benefits to developers	Consider the introduction.	
ES 7				Ge	Default comparisons will allow the reduction of boilerplate code.	Reconsider default comparisons or at least the ==/!= part.	
ES 8		23.1.1 [container.node] and paragraphs relating to this in 23.1 [container].		Te	Node handles are an over-specified solution to the relatively simple problem of moving nodes between associative containers, which can be done with a more conservative interface similar to std::list::splice. There is a lack of consistency with std::list, where splicing and merging can be done but there is no node handle-based interface, yet lists are indeed node based, too. P00832 acknowledges the simpler solution (by Talbot) but dismisses it as it offered “no further advantages”: however, the further advantages or use cases node handles allegedly provide are not clear at all.	Remove the changes proposed in P00382 and settle on a more conservative interface akin to that of std::list.	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
US 1		[expr] (5) and other clauses		te	The recent revisions to the rules for expression evaluation order are proving to be far more contentious than anticipated, and seem to be adversely affecting consensus for adopting this Committee Draft as the next C++ standard. See P0145R3	See P0145R3	
US 2		[expr] (5) and other clauses amended by ISO/IEC TS 19717:2015		te	Independent of their applicability to Concepts, the *requires-clause* and *requires-expression* parts of the Concepts-Lite TS seem generally regarded as useful and uncontroversial C++ language features. Adopting these features now would reduce dissatisfaction with the absence of Concepts-Lite from the CD, and thereby improve consensus for its adoption.	Extract (from ISO/IEC TS 19717:2015) the wording that specifies the syntax and semantics of the *requires-clause* and *requires-expression* features. Amend this wording pursuant to relevant issues list resolutions and then apply the updated wording.	
US 3		[expr.ass] (5.18) and/or other clauses affected by **P0145R3**		te	It is very surprising that expressions such as the following are required to have different outcomes when the evaluations of a and b have overlapping side effects: • a @= b • a.operator@=(b)	Ensure that such expression pairs are guaranteed to provide identical results and side effects. • Perhaps the simplest way to do so is to change in ¶1: “The ~~right~~ left operand is sequenced before the ~~left~~ right operand.” • Alternatively, restore the status quo ante.	
US 4		[dcl. decomp] (8.5)	¶3	ed	When referring to a type trait’s value, the _v forms are usually preferred.	Replace std::tuple_size<E>::value by std::tuple_size_v<E>.	
US 5		[over.binary] (13.5.2)	¶1	te	Remove users’ need to write boilerplate code for many or most of the comparison operators !=, >, <=, and >=, while: • Preserving backward compatibility for the Standard Library as well as for all existing well-formed user code, and • Remaining faithful to the *EqualityComparable* and *LessThanComparable* concepts (as promulgated, for example, in SGI’s implementation of the STL).	Append to ¶1 (or add as new ¶2): If neither form of the operator function has been declared, then for each binary operator @ appearing in the left column of Table *n*, x @ y shall instead be reinterpreted as shown in the corresponding right column entry. Table *n* — Reinterpretation of selected binary operators [reinterpretation] Expression — Reinterpretation x != y — !(x == y) x > y — y < x x >= y — !(x < y) x <= y — !(y < x)	
US 6		[temp.deduct] (14.8.2)		te	Per [c++std-core-26539], “we're missing the core wording for template argument deduction for partial	Provide the missing wording, thereby possibly also resolving related open CWG issues such as 697 and	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					specializations." This lack affects such code as the detection idiom's application of void_t, as exemplified in the Library Fundamentals 2 TS.	2054.	
US 7		All library Clauses		te	P0091R3 "Template argument deduction for class templates (Rev. 6)" was adopted for the core language, but the **Standard Library makes no explicit use of this new feature**, even though the promise of such use provided strong motivation for the feature.	Analyze the Standard Library's constructors to determine which classes would profit from explicit deduction guides. Formulate the appropriate guides for those classes and insert them in their respective types.	
US 8		All library Clauses		te	The Standard Library mistakenly uses *Requires:* clauses to express two distinct kinds of requirements: some requirements can be statically checked, while others can't. We should insist on statically checked requirements wherever possible, leading to an ill-formed program when such a requirement is violated.	**See p0411r0**	
US 9		[meta.type. synop] (20.15.2)	Synopsis	ed	Unlike all other value-returning type traits, this synopsis has no entry for has_unique_object_representations_v. See also the related comment re [meta.unary.prop] (20.15.4.3).	Insert the missing entry, with the obvious definition, following the entry for has_virtual_destructor_v.	
US 10		[meta.type. synop] (20.15.2)	¶1	te	A user specialization of any type trait should produce an ill-formed program, not merely one whose behavior is unspecified. See also the related comment re [execpol. type] (20.19.3).	Reword the paragraph as follows: Unless otherwise specified, a program that adds specializations for any of the templates defined in this subclause is ill-formed; no diagnostic required.	
US 11		[meta.unary.prop] (20.15.4.3)	Last row of Table 38 and also ¶9	ed	For consistency with similar specifications, has_unique_object_representations_v<T> should be used in place of has_unique_object_representations<T> ::value. See also the related comment re [meta.type.synop] (20.15.2).	Make the obvious replacements.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 12		[meta.unary.prop] (20.15.4.3)	Table 38	ed	The conditions for is_signed and is_unsigned unnecessarily refer to bool_constant.	Remove bool_constant<>::value from these two entries, leaving only the boolean expressions that these tokens surround.	
US 13		[meta.unary.prop] (20.15.4.3)	Table 38	ed	When referring to a type trait's value, the _v forms are usually preferred.	Replace std::is_destructible<T>::value by std::is_destructible_v<T> throughout the affected table cell.	
US 14		[execpol. type] (20.19.3)	¶3	te	A user specialization of any type trait should produce an ill-formed program, not merely one whose behavior is unspecified. See also the related comment re [meta.type. synop] (20.15.2).	Reword the paragraph as follows: Unless otherwise specified, a program that adds specializations for is_execution_policy is ill-formed; no diagnostic required.	
US 15		25.2.4	2	te	Calling 'std::terminate' when an element access function exits via. an uncaught exception effectively disables the normal means of C++ error handling and propagation when using the parallel algorithms. This will be both confusing to users and a common source of bugs. Furthermore, by defining this behavior we are essentially preventing further solutions to this problem.	There are several solutions that would be acceptable, among them: 1. Make it undefined behavior when an element access function exits via. an uncaught exception. This will allow for a future solution to this problem that is backwards compatible. 2. When an element access function exits via. an uncaught exception, throw a 'std::exception_list' which represents a collection of exceptions that were thrown in parallel. 3. When an element access function exits via. an uncaught exception, throw an unspecified 'std::exception'. 4. Rename the parallel algorithms to clarify that exception throwing code will result in a call to 'std::terminate'. For example 'std::exceution::parallel_policy' would be renamed to 'std::exceution::parallel_policy_noexcept' and 'std::execution::par' would be renamed to 'std::execution::par_noexcept'.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 16		25.2.5	2	te	It is unclear what behavior a parallel algorithm will have when a user-provided function exits via. an uncaught exception. This statement seems to require most parallel algorithms to nodeterministically choose one of the exceptions thrown and then re-throw that in the calling thread.	Clarify in section 25.2.5 what happens when a user-provided function throws an exception.	
US 17		25.2.5	2	te	This statement seems to require most parallel algorithms to nodeterministically choose one of the exceptions thrown and then rethrow that in the calling thread. In the case that multiple threads witness an exception from a user-provided function, all but one of those exceptions gets discarded. It is much preferrable to have all exception data preserved.	When a user-provided function exits via. an uncaught exception, throw a 'std::exception_list' structure which represents a collection of exceptions that were thrown in parallel.	
US 18		[depr.except.spec] (D.3) and other subclauses per P0003r4		te	Dynamic exception specifications have long been superseded, and are widely regarded as having been a mistake. They have previously been deprecated; it's time to excise them.	Apply the proposed wording from **p0003r4**	

US 19		13.3.1.8, 14.9 and Clauses 17-30 (all library clauses)		te	The Standard Library should be reviewed with the purpose of ensuring it takes proper advantage of template deduction for constructors.	• Review all classes in the standard library. For some classes, no changes may be required: std::complex c(2.1, 3.5); // Deduce complex<double> by 14.9 In other cases, explicit deduction guides may be necessary
						
						int i{5}; std::tuple c(2.1, reference_wrapper(i)); // Seems like it should behave like make_tuple
						
						The review should also consider whether constructors in the standard library create too much ambiguity, making it impossible even with explicit guides to deduce the parameters. If this happens, options such as the following could be considered	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						1. Making it possible to remove an implicit guide from the overload set 2. Giving explicit guides precedence over implicitly deduced guides 3. Removing implicit guides from C++17	

US 20		13.3.1.8, 14.9		TE	As pointed out in **P0091R3**, T&& arguments in constructors traditionally refer to rvalue references.
					
					template<class T> struct Wrapper
					{
					T value;
					Wrapper(T const& x): value(x) {}
					Wrapper(T && y): value(std::move(x)) {} // intent is rvalue reference
					};
					int main() {
					std::string foo = "Hello";
					auto w = Wrapper(foo); // Error. Universal reference is deduced
					}
					
					While P0091R3 proposes that such cases can be handled with explicit deduction guides, a more transparent solution would be desirable
						As an alternative to the approach in **P0091R3**, consider whether implicit deduction guides should use SFINAE to constrain to rvalue references like was intended in the constructor.	

US 21				te	The “operator dot” functionality is missing from the CD. It has been widely expected to be included in this version of the standards.	Integrate the functionality as described in the latest versions of **P0416r0 and P0252r1**	
US 22				te	The “std::byte” paper was reviewed and approved by EWG for C++17. Its integration is missing from the CD because it is awaiting a final review by LWG. This feature increases type safety in C++.	**See p0298r1** **See p0137r1**	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
US 23		8.5	1	te	The “structured bindings” proposal originally used braces “{}” to delimit binding identifiers. Those delimiters were changed to brackets “[]” under the assertion that they didn’t introduce any syntactic problem. However, they turned out to introduce syntactic ambiguity with attributes and lambdas. In the light of various suggested fixes, it appears the original syntax is more adequate.	Change the delimiters to curly braces.	

US 24		9.2.3.2	3	te	The current specification prohibits constexpr static data members that are of the same type as the enclosing class. Example:
					struct A {
					  int val;
					  static constexpr A cst = { 42 }; // error
					};
					
					int main() {
					  Return A::cst.val;
					}
						Defer semantics processing of initializers of constexpr static data members until the completion of the scope of the enclosing class. Effectively allowing this construct.	

US 25		27.10.8.4.10	7	te	has_filename() is equivalent to just !empty(). (So remove_filename() fails its postcondition in its examples.) The current definition of the relevant predicate is useless and (therefore) ignored by the functions that mention it.	Remove it, or reconsider after adjustments to definition of filename() and remove_filename() already discussed.	
US 26		12.1	4	ed	"either has no parameters" is (technically) redundant	Rephrase as a parenthetical after the general case.	
US 27		12.6.2	10	ed	“side effects” in the example	Remove space.	
US 28		15.2	4	te	depends on “principal constructor” being the innermost one (the non-delegating constructor), but §12.6.2¶6 defines “principal constructor” as the outermost one (the non-target constructor)	Change the definition in §12.6.2¶6 to be the non-delegating constructor.	
US 29		20.8.3	2	te	What does it mean for (the contained) objects to be “equivalent”?	Add definition (note that using `operator==()` involves complicated questions of overload resolution).	
US 30		26.8.7	2	ge	It is highly unusual that the value of (what is for	Call attention to the peculiarity (which can be useful	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					random access iterators) `last-1` is unused; this prohibits usage of an entire container (since `end()+1` is UB).	when the input iterators are not bidirectional). Provide *also* the scan from Scala, where the output range is one longer than the input.	
US 31		27.10		ge	It is unfortunate that everything is defined in terms of one implicit host system (cf. Python's `posixpath`, that can be imported anywhere); consider, for example, the impediment to a test suite.	Possibly: add a template argument for selecting the syntax, with (at least) POSIX and Windows conventions defined.	
US 32		27.10.2.1	3	ge	What does it mean to not "provide behavior that is not supported by a particular file system"? (Is it permissible for the functions to not exist at all on an implementation that expects to operate only with such a file system?)	Clarify that ¶2 governs and an error must be reported in such cases.	
US 33		27.10.4.2		ge	This definition is problematic: it is time-dependent, needs permissions to verify, and conflicts with "normal form" because it prohibits dot elements.	Remove entirely, since it is unused.	
US 34		27.10.4.5		ge	Are there attributes of a file that are not an aspect of the file system?	State that all are included, or give examples of those that may not be.	
US 35		27.10.4.6		te	What synchronization is required to avoid a file system race? For many systems, the file system itself is an important means of synchronization; if that is not permitted, the entirety of §27.10 is useless for many applications.	Specify the synchronization requirements, perhaps the very weak ones from POSIX: If a *read()* of file data can be proven (by any means) to occur after a *write*() of the data, it must reflect that *write*(), even if the calls are made by different processes.	
US 36		27.10.4.9		ge	Symbolic links themselves are attached to a directory via (hard) links.	Correct definitions; allow creating hard links "to" (really "for") symbolic links in §27.10.15.3¶3.4.3.	
US 37		27.10.4.12		ge	The term "redundant current directory (*dot*) elements" is not defined.	Define it as, presumably, any dot element except the special case of having one at the end as a directory name marker.	
US 38		27.10.4.13		ed	duplicates §17.3.16	Remove.	
US 39		27.10.4.15	(the note)	ed	dot and dot-dot are not directories (merely aliases for some directory), so it is meaningless to say they have no parent.	Remove the note.	
US 40		27.10.4.15		ge	Not all directories have a parent.	Mention this, and perhaps cross-reference	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
						§27.10.8.1¶2 about `/..`.	
US 41		27.10.4.16		ed	The term "parent directory" for a (non-directory) file is unusual.	Use "containing directory" instead, perhaps in §27.10.4.15 as well.	
US 42		27.10.4.21		ed	Pathname resolution does not always resolve a symlink.	State this.	
US 43		27.10.5	4	ge	The "encoded character type" idea suggests that paths are the result of encoding some character sequence. Unfortunately, this is often untrue in practice: Windows implementations typically use a 16-bit `wchar_t` that, in violation of §3.9.1¶5, is not actually a character but a two-byte unit that nominally stores results from the UTF-16 encoding but is actually uninterpreted (significant for surrogate pairs). Similarly, typical Linux implementations use 8-bit char in expectation of, but without requiring, UTF-8 encoding. Directory separators are recognized directly from these non-character representations, so it is appropriate for applications to work directly with the sequences of byte or two-byte units and perform decoding as a further step if desired.	Remove suggestion that applications may rely on decoding a `path` into a sequence of characters, and that the exclusion of `signed char` and `unsigned char` results from their failure to be an encoding of anything. Warn for functions like `path::string()` that the conversion may fail.	
US 44		27.10.8		te	The explicit definition of `path` in terms of a string requires that the abstraction be leaky. Consider that the meaning of the expression `p+='/'` has very different behavior in the case that `p` is empty; that a path can uselessly contain null characters; and that iterators must be constant to avoid having to reshuffle the packed string.	Define member functions to express a `path` as a string, but define its state in terms of the abstract sequence of components (including the leading special components) already described by the iterator interface. Remove members that rely on arbitrary manipulation of a string value.	
US 45		27.10.8.1		ge	The portability of the generic format is compromised by the unspecified *root-name*.	Place limits on the contents of a *root-name*, or dispense with the generic format entirely in the course of addressing the previous issue by weakening the `path`-string connections.	
US 46		27.10.8.1		ge	*filename* can be empty, so the productions for *relative-path* are redundant.	Simplify the grammar: perhaps drastically, since any string matches by some sequence of *name* and *directory-separator* productions.	
US 47		27.10.8.1		ed	"." and ".." already match the *name* production.	Exclude them from it, or else remove the *filename*/*name* distinction.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
US 48		27.10.8.1	1	ge	Multiple separators are often meaningful in a *root-name*.	Limit the scope of the paragraph to the *relative-path*.	
US 49		27.10.8.2.2	1.3, 1.4	ge	What does “method of conversion method” mean?	Reword.	
US 50		27.10.8.3	1.4	ed	largely redundant with ¶1.3	Remove; add “that after array-to-pointer decay” and `decay_t<Source>` to ¶1.3.	
US 51		27.10.8.4.3	2.3	te	Failing to add a / when appending the empty string constitutes a discontinuity (in the length of the output as a function of the length of the inputs) and prevents useful applications like forcing a symlink to be resolved.	Follow the example of Python’s `path.join()`.	
US 52		27.10.8.4.5	5	te	The postcondition is not by itself a definition, as illustrated by the non-idempotent behaviour in the example.	Add a definition.	
US 53		27.10.8.4.5	7	te	The “example behavior” does not correspond to the function name, which suggests `/foo/bar` → `/foo/` → `/foo/`.	Rename the function to `remove_component()`, or alter it to follow Python’s `path.dirname()` (including its treatment of `/`).	
US 54		27.10.8.4.5	10	te	The example demonstrates that this function is broken (perhaps because the underspecified `remove_filename()` is not the right thing). The undesirable discontinuity of `operator/=()` is also inherited.	Define in terms of improved and clarified versions of the underlying functions.	
US 55		27.10.8.4.5	11	ge	This is the most egregious example (among many) of using the type `path` inappropriately: `replacement` is a string, not a `path` that might include things like roots.	Use `string_type` for this and similar parameters.	
US 56		27.10.8.4.5	11.2	ge	The conditional addition of the period produces a(nother) discontinuity; applications will have to include the period anyway to support empty extensions.	Never add a period.	
US 57		27.10.8.4.8	2	ge	On Windows, absolute paths will sort in among relative paths.	Consider including the absoluteness of a path in its sort key.	
US 58		27.10.8.4.9	5	te	The behavior for root paths is useless: “/” becomes “” and (on Windows) “c:\\” becomes “c:” which is in no	Follow Python’s `path.dirname()`. If the purely component-based definition is desired, give it a name like `most components()` (inspired by the	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					way a parent of it.	Wolfram Language).	
US 59		27.10.8.4.9	6	te	Again, using `path` for single path components is bizarre.	Return `string_type` from this and other similar functions (not including `root_name()` and `root_path()`, which make sense as `path`s).	
US 60		27.10.8.4.9	6	te	`path("/foo/").filename()==path(".")` is surprising.	Follow Python's `path.basename()` and return an empty `string_type`.	
US 61		27.10.8.4.9	8	te	Leading dots in `filename()` should not be taken to begin an extension (*e.g.*, `.bashrc`).	Follow Python's `path.splitext()` in ignoring them.	
US 62		27.10.8.4.9	11	te	It is important that `stem()+extension()==filename()`.	Require implementations to preserve this.	
US 63		27.10.8.4.11	1	ge	It is inconsistent to take a trailing `/` as indicative of a directory but not a trailing `/..`, (which must refer to one).	Append the `/.` in all cases known to name directories (if it is in fact necessary).	
US 64	all	all	all	ge	The present references to UCS2 in the Committee Draft are appropriate in the interests of preventing silent breakage of software written to older versions of C++.	Preserve the references to UCS2 as presented in the Committee Draft.	
US 65	all	all	all	ge	The adoption of the changes proposed in WG21 document **P0386R2** (inline variables) is a step in the right direction.	Preserve the functionality as presented in the Committee Draft.	
US 66	all	all	all	ge	The adoption of the changes proposed in WG21 document **P0292R2** (constexpr if-statements) is a step in the right direction.	Preserve the functionality as presented in the Committee Draft.	
US 67	all	all	all	ge	Further consideration of the proposal known as Operator Dot (in **P0416R0**, its predecessors, etc.) for incorporation into the current new revision of IS 14882 is not desired. The topic was controversial among the experts in WG21. The C++ community will benefit if the feature is not rushed.	Limit the adoption of Operator Dot such that it may only be incorporated in a later revision of 14882 (not the revision of 14882 for which SC22 N5131 is a Committee Draft ballot).	
US 68	all	all	all	ge	Further consideration of the proposal known as Unified Call Syntax (in **P0301R1**, its predecessors, etc.) for incorporation into the current new revision of IS 14882 is not desired. The topic was controversial among the experts in WG21. The C++ community will	Limit the adoption of Unified Call Syntax such that it may only be incorporated in a later revision of 14882 (not the revision of 14882 for which SC22 N5131 is a Committee Draft ballot).	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					benefit if the feature is not rushed.		
US 69	all	all	all	ge	Further consideration of the proposal known as Default Comparisons (in **P0221R2**, its predecessors, etc.) for incorporation into the current new revision of IS 14882 is not desired. The topic was controversial among the experts in WG21. The C++ community will benefit if the feature is not rushed.	Limit the adoption of Default Comparisons such that it may only be incorporated in a later revision of 14882 (not the revision of 14882 for which SC22 N5131 is a Committee Draft ballot).	
US 70	all	all	all	te	The adoption of **P0003R4** (Removing Deprecated Exception Specifications) would reduce language complexity and resolve all specification issues related to its presence in the IS.	**Adopt P0003R4.**	
US 71	all	7 [dcl.dcl]	paragraph 1	te	The [*identifier-list*] syntax for decomposition declarations has been reviewed for grammar ambiguities, and is likely to be less problematic in the face of future evolution than the case where curly braces “{ }” are adopted in place of the square brackets.	Preserve the syntax of decomposition declarations as presented in the Committee Draft.	
US 72	all	1.8 [intro.object]	Para 3	te	The introduction of additional special behavior for unsigned char in contexts where it may already occur in programs today is harmful to the optimization which may be obtained.	Adopt std::byte (**P0257R1**) with necessary changes from WG21 review and modify 1.8 [intro.object] paragraph 3 by replacing “array of *N* unsigned char” with “array of *N* std::byte”.	
US 73	all	27.10.8.1 [path.generic]	all	te	*root-name* is effectively implementation defined. As acknowledged by the note under *root-name* in the grammar, // is an example of what a *root-name* may be. Should *root-name* be // for a specific implementation, the grammar is ambiguous. The string //a may resolve as either ***root-name*** *root-directory*$_{opt}$ *relative-path*$_{opt}$ //***root-directory***$_{opt}$ *relative-path*$_{opt}$ //***relative-path***$_{opt}$ //***filename***	Change under *root-name* in the grammar of subclause 27.10.8.1 [path.generic]: An **implementation defined path prefix** ~~operating system dependant name~~ that identifies the starting location for absolute paths. Add a new paragraph before paragraph 1 of [path.generic]: The *root-name* in a *pathname* is the longest sequence of characters that could possibly form a *root-name*.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					//***name*** //***a*** or ***root-directory*** *relative-path*$_{opt}$ ***directory-separator*** *relative-path*$_{opt}$ ***slash directory-separator*** *relative-path*$_{opt}$ ***slash*** *directory-separator relative-path*$_{opt}$ ***/directory-separator*** *relative-path*$_{opt}$ /***slash*** *relative-path*$_{opt}$ /***/relative-path***$_{opt}$ //***filename*** //***name*** //***a***		
US 74	all	27.10.8 [class.path]	all	te	The term “pathname” in 27.10.8 [class.path] is ambiguous in some contexts.	Add the following specification to 27.10.8.2.1 [path.fmt.cvt]: Specifications for path appends, path concatenation, path modifiers, path decomposition and path query are in terms of the generic pathname format. An implementation needs to make whatever changes necessary to the pathname in native pathname format to produce the specified change in the generic pathname format, or return query result for pathname in terms of the generic pathname format. **See p0430r0 Section 2.1**	
US 75	all	27.10.8.4.1 [path.construct]	all	te	Extra flag in path constructors is needed to distinguish whether source is in native pathname format, or generic pathname format.	Refer to **P0430R0** section 2.2	
US 76	all	27.10.8.1	all	te	*root-name* definition is over-specified.	See p0430r0 section 2.3.1	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
		[path.generic]			The description of *root-name* limits its use to be the starting location for absolute paths. This is overly restrictive and disregards established practice where special prefixes on path names is treated as a trigger for alternate path resolution on certain operating systems. There are cases where such alternative path resolution relies on context from the environment such as the identity of the current user; therefore, the presence of a special prefix on a path name is not always indicative of an absolute path.		
US 77	all	27.10.8.4.3 [path.append]	all	te	operator/ (and other append) semantics not useful if argument has *root-name*. A non-POSIX operating system could design its generic pathname for native file type to have a *root-name* and use it in some creative way. For example, if argument p has a *root-name*, then p's *root-name* have to be removed before appending.	See **p0430r0** section 2.3.2.	
US 78	all	27.10.15.1 [fs.op.absolute]	all	te	Member function absolute in 27.10.4.1 is over-specified for non-POSIX-like operating system. .	See p0430r0 Section 2.4.1	
US 79	all	27.10.13 [class.directory_iterator] 27.10.15.3 [fs.op.copy] 27.10.15.14 [fs.op.file_size] 27.10.15.35 [fs.op.status]	all	te	Some file system operation functions are over-specified for implementation-defined file type.	See **p0430r0** section 2.4.2	
US 80		21.4		**te**	Missing basic_string_view literals	We have “”s for string literals, but nothing to create string_views. Add similar wording as in [basic.string.literals], but for basic_string_view, preferably using “”sv . And they should be constexpr.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
US 81		21.2.3.x		**te**	More char_traits member functions should be constexpr	With string_view, we can now build more things at compile time. However, char_traits is limiting us here. Mark more of the member functions in char_traits as constexpr (in particular, compare, length and find). The member functions move, copy and pointer-based assign need not be constexpr, but everything else should be.	
US 82		Entire draft		ge	Address existing open issues in core and library issues lists	Make technical and editorial changes as appropriate for each issue, or resolve as NAD	
US 83		16.8	¶ 1	te	The definition of the macro __cplusplus refers to C++14, not C++17	Update definition to reflect the expected ratification month	
US 84		20.14.2	¶ 2	te	The distinction between *INVOKE*(f, t1, t2, … tN) and *INVOKE*(f, t1, t2, … tN, *R*) is too subtle. If the last argument is an expression, it represents tN, if it's a type, then it represents R. Very clumsy.	Rename *INVOKE*(f, t1, t2, … tN, *R*) to *INVOKE_R*(*R*, f, t1, t2, … tN) and adjust all uses of this form. (Approximately 10 occurrences of invoke would need to change.)	

US 85		20.15.2 and 20.15.6		te	The trick of encoding a functor and argument types as a function signature for is_callable and result_of loses cv information on argument types, fails for non-decayed function types, and is confusing. E.g.,
					typedef int MyClass::*mp;
					result_of_t<mp(const MyClass)>;
					// should be const, but isn't
					typedef int F(double);
					is_callable<F(float)>; *// ill-formed*
						Minimal change:
						Replace
						is_callable<Fn(ArgTypes...)>
						with
						is_callable<Fn, ArgTypes...>
						and replace is_callable<Fn(ArgTypes...), R>
						with is_callable_r<R, Fn, ArgTypes...>.
						Do the same for is_nothrow_callable
						
						Preferred change: All of the above, plus deprecate
						result_of<Fn(ArgTypes...)>
						and replace it with
						result_of_invoke<Fn, ArgTypes...>	

US 86		20.15.2 and 20.15.6		te	"is_callable" is not a good name because it implies	Rename "is_callable" to "is_invocable" and rename	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					F(A…) instead of *INVOKE*(F, A…)	"is_nothrow_callable" to "is_nothrow_invocable"	
US 87		1.10.2	¶ 14	ed	The term "block with forward progress guarantee delegation" is cumbersome. "Forward" is redundant and "guarantee" is implicit.	Replace the term "block with forward progress guarantee delegation" with "block with progress delegation" throughout the standard.	
US 88		20.19.4	Section heading	ed	"Sequential" should be "Sequenced" (per **P0336r1**, which was adopted 2016-06)	Change "Sequential" to "Sequenced" in section heading	
US 89		20.19.6	Section heading	ed	"Parallel+Vector" should be "Parallel+Unsequenced" (per **P0336r1**, which was adopted 2016-06)	Change "Parallel+Vector" to "Parallel+Unsequenced" in section heading and change section label from "[execpol.vec]" to "[execpol.parunseq]"	
US 90		25.2.3	¶ 1	ed	Need a cross-reference directing readers to execution policies [execpol] section	Add a cross-reference link to section 20.19, somewhere within the paragraph.	
US 91		25.3, 25.4, 25.5		ed	Presentation of parallel algorithms is confusing. Despite having parallel overload prototypes in section 25.1 <algorithm> synopsis and blanket wording 25.2.5, it is still confusing to figure out which algorithms have parallel overloads.	Copy the prototypes for the parallel algorithm overloads alongside their serial versions in the per-algorithm description. The common description of a serial and parallel overload will reinforce that they exist and have the same semantics. In the cases where they do not have the same semantics, their separate descriptions will make that clear, too.	
US 92		5.1.5 [expr.prim.lambda]	1	Te	Lambda *init-capture*s should support some form of decomposition declaration, as functions returning values intended for decomposition will become a much more common idiom.	Amend the *init-capture* grammar to allow for a decomposition-capture.	
US 93		5.2.2 [expr.call]	5	Te	It is not immediately clear that expressions in the *expression-list* will have a fully-specified order of evaluation if the called function is an overloaded operator.	Add a second note to **5.2.2 [expr.call]** p5 with a cross-reference to **13.3.1.2 [over.match.oper]** clarifying that the *expression-list* is evaluated in a fully specified order when the function call is an overloaded operator – ideally by providing an example.	
US 94		5.2.3 [expr.type.conv]	2	Te	To properly support universal initialization syntax with class template deduction, this paragraph should	Duplicate the wording for T(x1, x2, ...) to also handle T{x1, x2, ...}	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					support initialization through T{x1, x2, ...} as well as through T(x1, x2, ...). It is expected that while aggregates would not implicitly be deduced this way, a deduction guide should be able to offer such support where desired.		
US 95		7 [dcl.dcl]	8	Te	There is no obvious reason why decomposition declarations cannot be declared as static, thread_local, or constexpr.	Allow constexpr, static, and thread_local to the permitted set of *decl-specifiers*.	
US 96		8.5 [dcl.decomp]		Ed	This specification would read much more easily with the usual 0-based indexing than the current 1-based index.	Use 0-based indexing for the identifier-list, and replace all use of 'i-1' with just 'i'. The existing 'i' subscripts would not need to change for this rebasing.	
US 97		8.5 [dcl.decomp]	3	Ed	Prefer to use tuple_size_v and tuple_element_t consistently through the standard, than the more verbose tuple_size<E>::value and tuple_element<i-1, E>::type	Consistently use _v/_t form for type traits.	
US 98		8.5 [dcl.decomp]	3	Te	The lifetime-extension rules when binding a reference to a temporary do not seem to apply to: auto [x,y] = std::make_pair<std::string, string>("hello", "world");	Address the issue of lifetime extension when a decomposition declaration potentially binds a reference to a temporary object.	
US 99		8.5 [dcl.decomp]		Ge	Decomposition declarations are confusing in generic code: auto [x,y,z] = f(a,b,c); may bind references if the result is a pair or tuple (returned by value); or copy distinct objects if f returns an array by reference, or returns an aggregate (by value or by reference).	Provide more consistent semantics for predictable behavior within function templates by not implicitly binding references to results returned by value, or by always binding references (and extending lifetimes) in such cases.	
US 100		8.5 [dcl.decomp]		Ge	Decomposition declarations should provide syntax to discard some of the returned values, just as std::tie	Extend the grammar of decomposition declarations to support discarded values, such as by allowing	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					uses std::ignore.	void in the *identifier-list*.	
US 101		9 [class]	10	Ge	The term POD no longer serves a purpose in the standard, it is merely defined, and restrictions apply for when a few other types preserve this vestigial property. The is_pod trait should be deprecated, moving the definition of a POD type alongside the trait in Annex D, and any remaining wording referring to POD should be struck, or revised to clearly state intent (usually triviality) without mentioning PODs.	Move the definition of is_pod/is_pod_v to **D.12 [depr,meta.types]** Move **9p10 [class]** into **D.12 [depr,meta.types]** Reword footnote 40 in terms of trivial constructors Strike POD classes and the definition of POD types from **3.9p9 [basic.types]** Strike **5.1.5 [expr.prim.lambda]** p4 bullet 4.4 Strike footnote 108 (from 9p10) Strike the reference to POD type in **17.3.4 [defns.character.container]** Revise definition of max_align_t in **18.2.3 [support.types.layout]** p5 Revise definition of aligned_storage::type in table 46 - Other transformations Revise definition of aligned_union::type in table 46 - Other transformations	

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

ISO/IEC/CEN/CENELEC electronic balloting commenting template/version 2012-03

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						Update the introductory sentence to **21.1[strings]** p1	
US 102		13.3.1.2 [over.match.oper]	2	Te	It is no longer legal to manually transform code from infix form to function form. For example, the expression a() = b() sequences b() before a() while a().operator=(b()) sequences a() before b().	Require a left-to-right order of evaluation for assignment operators, and for compound-assignment operators, consistent with such requirements on other operators.	
US 103		14.9 [temp.deduct.guide]	2	Te	It is not clear that when a *simple-template-id* names a template specialization, the default template parameters of the primary template by still be relied upon. The example from **p0091r3** that clearly shows this is the intent: template <class Iter> vector(Iter b, Iter e) -> vector<typename iterator_traits<Iter>::value_type>; The allocator of the vector is clearly not named, and expected to deduce as the default allocator (std::allocator< typename iterator_traits<Iter>::value_type>).	If the wording is already thought to state this clearly enough, add an example (such as in this comment) to clarify intent for the reader. Otherwise, amend the wording as necessary so that default template arguments will be used, as needed, to fill out the name of the class template specialization.	
US 104		16.1 [cpp.cond]		Te	__has_include has an ugly __ prefix that is not connected to a joining symbol. This appears necessary to avoid intruding on user-defined macros, but there are alternative solutions. For example, a '__' anywhere in a name is reserved to the implementation, so we could put the '__' in the middle instead,	Replace all use of __has_include with has__include	
US 105		17-30 plus Annex D		Ge	The library has been getting more careful about specifying runtime preconditions and constraints in	Adopt a revision of **p0411r0**	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					the type system, but both are documented in the same *Requires* clause which often could be clearer, especially when constraining how function templates interact with SFINAE. The terminology should be made more precise, with an expectation to uncover and clean up a few surprising corner cases as part of the process.		
US 106		17-30 plus Annex D		Ge	Review the whole library for constructors using member typedefs to name constructor parameters rather than template type parameters, as this inhibits class template deduction. e.g., the unique_lock explicit constructor taking the mutex_type typedef would be better served naming Mutex directly, to preserve support for deduction.	Review each constructor of each library class template, and revise specification of parameter types as needed.	
US 107		17.3 [defintions]		Te	The term 'direct non-list initialization' needs to be incorporated from the Library Fundamentals TS, as several components added to C++17 rely on this definition.	Add: **17.3.X direct-non-list-initialization [defns.direct-non-list-init]** A direct-initialization that is not list-initialization.	
US 108		20.2.2 [utility.swap]		Te	swap is a critical function in the standard library, and should be declared constexpr to support more widespread support for constexpr in libraries. This was proposed in **p0202r1** which was reviewed favourably at Oulu, but the widespread changes to the <algorithm> header were too risky and unproven for C++17. We should not lose constexpr support for the much simpler (and more important) <utility> functions because they were attached to a larger paper. Similarly, the fundamental value wrappers,	Adopt the changes to the <utility> header proposed in **p0202r1**, i.e., only bullets C, D, and E. In addition, mark the swap functions of pair and tuple as constexpr, and consider doing the same for optional and variant.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					pair and tuple, should have constexpr swap functions, and the same should be considered for optional and variant. It is not possible to mark swap for std::array as constexpr without adopting the rest of the **p0202r1** though, or rewriting the specification for array swap to not use swap_ranges.		
US 109		20.5.1 [tuple.general]		Te	tuple should be a literal type if its elements are literal types; it fails because the destructor is not necessarily trivial. It should follow the form of optional and variant, and mandate a trivial destructor if all types in Types... have a trivial destructor. It is not clear if pair has the same issue, as pair specifies data members first and second, and appears to have an implicitly declared and defined destructor.	Document the destructor for tuple, and mandate that it is trivial if each of the elements in the tuple has a trivial destructor. Consider whether the same specification is needed for pair.	
US 110		20.5.2.1 20.6.3.1 20.11.1.2.1		Te	The move constructors for tuple, optional, and unique_ptr should return false for is_(nothrow_)move_constructible_v<TYPE> when their corresponding *Requires* clauses are not satisfied, as there are now several library clauses that are defined in terms of these traits. The same concern applies to the move-assignment operator. Note that pair and variant already satisfy this constraint.		

US 111		20.6.3.1 [optional.object]		Te	The copy and move constructors of optional are not constexpr. However, the constructors taking a const T& or T&& *are* constexpr, and there is a precedent for having a constexpr copy constructor in **26.5.2 [complex]**. The defaulted copy and move	Add constexpr to:
						constexpr optional(const optional &);
						constexpr optional(optional &&) noexcept(*see below*);	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					constructors of pair and tuple are also conditionally constexpr (see **20.4.2 [pairs.pair]** p2 and **20.5.2.1 [tuple.cnstr]** p2). A strong motivating use-case is constexpr functions returning optional values. This issue was discovered while working on a library making heavy use of such.		
US 112		20.7.2 [variant.variant]		Te	Variants with an empty set of alternatives fail to work for a number of reasons. This should be explicitly acknowledged in the design, lest we attract defect reports on those many failings.	Either add an explicit requirement that sizeof...(Types) > 0, or add a note that we believe this is already implicit in the specification that follows.	
US 113		20.7.2 [variant.variant]		Te	Variants cannot properly support allocators, as any assignment of a subsequent value throws away the allocator used at construction. This is not an easy problem to solve, so variant would be better served dropping the illusion of allocator support for now, leaving open the possibility to provide proper support once the problems are fully understood.	Strike the 8 allocator aware constructor overloads from the class definition, and strike **20.7.2.1 [variant.ctor]** p34/35. Strike clause **20.7.12 [variant.traits]** Strike the specialization of ~~uses_allocator~~ for variant in the <variant> header synopsis, **20.7.1 [variant.general]**.	
US 114		20.7.2 [variant.variant]	2	Te	variant needs to know the size of an object in order to compute the size of its internal buffer, so require that any cv-qualified object type in Types... be a complete type.	Add 'complete' in p2: "All types in Types... shall be (possibly cv-qualified) complete object types, (possibly cv-qualified) void, or references."	
US 115		20.7.2 [variant.variant]	2	Te	Support for void alternatives is confusing and underspecified; it should be deferred as an extension until a future standard. For example, if any of the alternatives is void, the current specification fails to satisfy the *Requires* clause for all 6 relational operators, and loses (shall not participate in overload	Strike '(possibly cv-qualified) void," from **20.7.2 [variant.variant]** p2 From **20.7.4 [variant.get]** Strike ", and T_I is not (possibly cv-qualified) void' from p3. Strike ", and T is not (possibly cv-qualified) void'	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					resolution) the copy constructor, move constructor, copy-assignment operator, move-assignment operator, swap member and free function. It is not clear that a variant with a void alternative can be visited, especially in the multiple-variant visitor case. Adding a void alternative will render an otherwise trivial variant destructor as non-trivial. Are all of these consequences the intended design?	from p5. Strike ", and T_I is not (possibly cv-qualified) void' from p7. Strike ", and T is not (possibly cv-qualified) void' from p9.	
US 116		20.7.2 [variant.variant]	2	Te	Support for array alternatives does not seem to work as expected. For example, if any of the alternatives is an array, the current specification fails to satisfy the Requires clause for all 6 relational operators, and loses (shall not participate in overload resolution) the copy constructor, move constructor, copy-assignment operator, move-assignment operator (although the swap functions will work correctly). It is difficult to activate an array alternative - to the best of my understanding, it must be emplaced with no arguments in order to value-initialize the array, and then the value of each element may be assigned as needed. Many of these issues would be resolved if array alternatives were implemented by storing a std::array instead, and then exposing the exposition-only array member (of the std::array) to the get functions, but that seems like an experimental change that should be investigated for the next standard. For C++17, we should drop support for arrays (but not std::array) as alternatives, in order to leave freedom	Add 'not an array' in p2: "All types in Types... shall be (possibly cv-qualified) object types that are not arrays, (possibly cv-qualified) void, or references to non-array objects."	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					to support them properly in the next standard.		
US 117		20.7.2 [variant.variant]	2	Ge	It is not clear what support is intended for function references. The presence of a function-reference in the list of alternatives causes some operations to fail to instantiate/exist at all, and there is no clear benefit to supporting function references but not function types.	Qualify references as 'references to object types': "All types in Types... shall be (possibly cv-qualified) object types, (possibly cv-qualified) void, or references to object types."	
US 118		20.7.2.1 [variant.ctor]	19, 23, 27, 31	Te	The form of initialization for the emplace-constructors is not specified. We are very clear to mandate "as if by direct non-list initialization' for each constructor in optional, so there is no ambiguity regarding parens vs. braces. That wording idiom should be followed by variant.	Insert the phrase "as if direct-non-list-initializing" at appropriate locations in paragraphs 19, 23, 27, and 31	
US 119		20.7.2.3 [variant.assign]		Te	The copy-assignment operator is very careful to not destroy the contained element until after a temporary has been constructed, which can be safely moved from. This makes the valueless_by_exception state extremely rare, by design. However, the same care and attention is not paid to the move-assignment operator, nor the assignment-from-deduced-value assignment template. This concern should be similarly important in these cases, especially the latter.		
US 120		20.7.4 [variant.get]	3,5	Ed	For void alternatives, the get functions returning a reference naturally fall out of overload resolution as you cannot make a reference to void, so there is no need to call out this special case. Note that this is	Strike ", and T_I is not (possibly cv-qualified) void' from p3. Strike ", and T is not (possibly cv-qualified) void' from p5.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					NOT the case for the get_if overloads, which would return a pointer to void.		
US 121		20.7.11 [variant.hash]	1	Te	The value of a variant comprises the index as well as the contained alternative (if any), as can be seen in the comparison operators. Make it clear that both parts should contribute to the hash result.	Add: [*Note:* The value of a variant comprises the active index and the currently contained value, if any. Both parts should contribute to the resulting hash value - *end note*]	
US 122		20.11.1.2.1 [unique.ptr.single.ctor]	4	Te	unique_ptr should not satisfy is_constructible_v<unique_ptr<T, D>> unless D is DefaultConstructible and not a pointer type. This is important for interactions with pair, tuple, and variant constructors that rely on the is_default_constructible trait.	Add a *Remarks:* clause to constrain the default constructor to not exist unless the *Requires* clause is satisfied.	
US 123		20.11.1.2.1 [unique.ptr.single.ctor]	12	Te	is_constructible_v<unique_ptr<P, D>, P, D const &> should be false when D is not copy constructible, and similarly for D&& when D is not move constructible. This could be achieved by the traditional 'does not participate in overload resolution' wording, or similar.	Add a *Remarks:* clause to constrain the appropriate constructors.	

US 124		20.11.2.2 [util.smartptr.shared]		Te	Several shared_ptr related functions have wide contracts and cannot throw, so should be marked unconditionally noexcept.	Add 'noexcept' to:
						
						template<class U> bool shared_ptr::owner_before(shared_ptr<U> const& b) const noexcept;
						template<class U>
						bool shared_ptr::owner_before(weak_ptr<U> const& b) const noexcept;
						
						template<class U> bool	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

						weak_ptr::owner_before(shared_ptr<U> const& b) const noexcept;
						template<class U>
						bool weak_ptr::owner_before(weak_ptr<U> const& b) const noexcept;
						
						bool owner_less::operator()(A,B) const noexcept; *// all versions*	

US 125		20.11.2.2.1 [util.smartptr.shared.const]	4	Te	This constructor should not participate in overload resolution unless the *Requires* clause is satisfied. Note that this would therefore apply to some assignment operator and reset overloads, via *Effects:* equivalent to some code wording.	Add a *Remarks:* clause to constrain this constructor not to participate in overload resolution unless the *Requires* clause is satisfied.	
US 126		20.11.2.2.1 [util.smartptr.shared.const]	8	Te	This constructor should not participate in overload resolution unless the *Requires* clause is satisfied. Note that this would therefore apply to some assignment operator and reset overloads, via *Effects:* equivalent to some code wording.	Add a *Remarks:* clause to constrain this constructor not to participate in overload resolution unless the *Requires* clause is satisfied.	
US 127		20.11.2.2.1 [util.smartptr.shared.const]	8	Te	It should suffice for the deleter D to be nothrow move-constructible. However, to avoid potentially leaking the pointer p if D is also copy-constructible when copying the argument by-value, we should continue to require the copy constructor does not throw if D is CopyConstructible.	Relax the requirement the D be CopyConstructible to simply require that D be MoveConstructible. Clarify the requirement that construction of any of the arguments passed by-value shall not throw exceptions. Note that we have library-wide wording in clause 17 that says any type supported by the library, not just this delete, shall not throw exceptions from its destructor, so that wording could be editorially removed. Similarly, the requirements that A shall be an allocator satisfy that neither	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
						constructor nor destructor for A can throw.	
US 128		20.11.2.2.1 [util.smartptr.shared.const]	9	Te	As this constructor is taking ownership of a new pointer, it should enable shared_from_this with p (unless p == 0). Note that making this an *Effect* here renders the additional enable shared_from_this for a released unique_ptr in p27 redundant.	Add to *Effects*: The first and second constructors enable shared_from_this with (T*)p.	
US 129		20.11.2.2.1 [util.smartptr.shared.const]	22	Te	This constructor should not participate in overload resolution unless the requirements are satisfied, in order to give correct results from the is_constructible trait.	Add a *Remarks:* clause to constrain this constructor not to participate in overload resolution unless the *Requires* clause is satisfied.	
US 130		20.11.2.2.1 [util.smartptr.shared.const]	26	Te	There is no ability to supply an allocator for the control block when constructing a shared_ptr from a unique_ptr. Note that no further shared_ptr constructors need an allocator, as they all have pre-existing control blocks that are shared, or already have the allocator overload.	Add an additional shared_ptr constructor, template <class Y, class D, class A> shared_ptr(unique_ptr<Y, D>&& r, A alloc), with the same semantics as the existing constructor taking a unique_ptr, but using the alloc argument to supply memory as required.	
US 131		20.11.2.2.1 [util.smartptr.shared.const]	27	Te	The constructor delegated to by a call to r.release is a deduction context, so unique_ptr<Y,D>::pointer must not only convert to T*, but also *unambiguously* satisfy the deduction context, or the effects clause should include an explicit cast to T*. Such casts must not throw exceptions, or else the released pointer will not have its deleter run.	Revise this paragraph: [Added two (T*) casts, added restrictions on throwing] *Effects*: If r.get() == nullptr, equivalent to shared_ptr(). Otherwise, if D is not a reference type, equivalent to shared_ptr((T*)r.release(), r.get_deleter()). Otherwise, equivalent to shared_ptr((T*)r.release(), ref(r.get_deleter())). Casts to T* must not throw exceptions; otherwise, if an exception is thrown, the constructor has no effect. If r.get() != nullptr, enables shared_from_this with the value that was returned by r.release().	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
US 132		20.11.2.2.1 [util.smartptr.shared.const]	9, 27	Te	As paragraphs 8-11 apply equally to the constructor taking a unique_ptr due to the *Effects:* equivalent to some code rules, there is a conflict between p9 saying d(p) is run if an exception is thrown, and p27 saying it shall have no effect.	Strike the penultimate sentence of p27, and implicitly require the unique_ptr is released and deleter run if an exception is thrown.	
US 133		20.11.2.2.1 [util.smartptr.shared.const]	27	Ed	With the revised definition of *enables shared_from_this with p* in p1, there is no need to check r.get() != nullptr. Further, paragraphs 8-11 apply equally to the unique_ptr constructor due to the *Effects:* equivalent to some code rules, and we do not want to enable twice, so the whole sentence is redundant.	Strike the last sentence, which begins with "If r.get() != nullptr,".	
US 134		20.11.2.2.2 [util.smartptr.shared.dest]	1	Te	The semantics for destroying the deleter and the control-block are unclear. In particular, it is not clear that we guarantee a lack of race conditions destroying the control-block and deleter. Possible race-free implementations might destroy the deleter after running d(p), and before giving up the weak reference held by this shared_ptr; running the destructor for 'd' only when the last weak_ptr is destroyed, potentially at a much later date, but ensuring that d(p) completes before the shared_ptr gives up its weak reference; making a copy of 'd' in the destructor before manipulating the weak count, and then using this copy to run 'd(p)', even while the control-block could be concurrently reclaimed with an expiring weak_ptr in another thread. Note that this	Clarify that the shared_ptr weak ownership of the control block is released at the end of the destructor, and not as the destructor begins. Otherwise, the deleter might be destroyed even before the destructor gets to move a copy to call safely.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					may be related to LWG #2751. (Also, see the note in 20.11.2.2.10p1 [util.smartptr.getdeleter])		
US 135		20.11.2.2.7 [util.smartptr.shared.cmp]	2	Te	The less-than operator for shared pointers compares only those combinations that can form a composite pointer type. With the C++17 wording for the diamond functor, less<>, we should be able to support comparison of a wider range of shared pointers, such that less<>::operator(shared_ptr<A>, shared_ptr<B>) is consistent with less<>::operator(A *, B *).	Replace less<V> with just less<>, and drop the reference to composite pointer types.	
US 136		20.11.2.2.9 [util.smartptr.shared.cast]	2, 6, 10	Ed	The returns clause for each cast mentions storing a copy of the cast pointer in the returned shared_ptr, unless the original pointer is *empty*. However, even in the case of the empty shared_ptr, we might store such a value to satisfy the post-condition, so saying this in two places is redundant and potentially contradictory. It suffices to say that each cast returns (when successful) a shared_ptr that shares ownership with the shared_ptr argument. Note that static_pointer_cast (and reinterpret_pointer_cast) could be further simplified as: *Effects:* equivalent to return shared_ptr<T>{r, static_cast<T*>(r.get())};	Strike the un-necessary reference to storing an object in the otherwise clause of each paragraph (deferring to the *Effects* clause): *Returns:* If r is *empty*, an *empty* shared_ptr<T>; otherwise, a shared_ptr<T> object that ~~stores static_cast<T*>(r.get()) and~~ *shares ownership* with r.	
US 137		20.11.2.2.9 [util.smartptr.shared.cast]	(6.2)	Te	It is intuitive, but not specified, that the empty pointer returned by a dynamic_pointer_cast should point to	Rephrase as: Otherwise, shared_ptr<T>().	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					null.		
US 138		20.14.2 [func.require]		Ed	The *INVOKE* protocol is used widely beyond just the <functional> sub-clause, and really belongs in the front matter of clause 17, taking the definitions of call wrappers and callable entities with it.	Move **20.14.1 [func.def] to 17.3 [definitions]**, and **20.14.2 [func.require] to 17.6 [requirements]**.	
US 139		20.14.3 [func.invoke]		Te	As the *INVOKE* protocol is used widely throughout the library, support for the invoke wrapper function belongs at the same level as move, forward, and swap. Note that as the invoke function has not yet been published in a standard, this is the last chance to cheaply make such a refactoring.	Move the invoke function template into the <utility> header. Move **20.14.3 [func.invoke]** into **20.2 [utility]**	
US 140		20.14.14 [unord.hash]	2	Te	Specializations of std::hash for arithmetic, pointer, and standard library types should not be allowed to throw. The constructors, assignment operators, and function call operator should all be marked as noexcept. It might be reasonable to consider making this a binding requirement on user specializations of the hash template as well (in p1) but that may be big a change to make at this stage.		
US 141		20.15 [meta]		Ge	The free-standing <type_traits> header, through the is_callable trait relying on the definition of *INVOKE*, has a dependency on reference_wrapper in the non-freestanding <functional> header.	Remove the dependency on reference_wrapper in INVOKE, either by generalizing the support it is trying to offer for all such wrapper types, or deferring *INVOKE* support for reference_wrapper until a better solution for the dependencies can be worked out.	
US 142		20.15.2 [meta.type.synop]		Te	An alias template using the new template template auto deduction would make integral_constant slightly	Add to the synopsis of <type_traits>: template <auto N>	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					easier to use.	using integer_constant = integral_constant<decltype(N), N>;	
US 143		20.15.4.3 [meta.unary.prop]	Table 38	Te	An is_aggregate type_trait is needed. The emplace idiom is now common throughout the library, but typically relies on direct non-list initalization, which does not work for aggregates. With a suitable type-trait, we could extend direct non-list-initlaization to perform aggregate-initalization on aggregate types.	Add a new row to Table 38: template <class T> struct is_aggregate; T is an aggregate type ([dcl.init.aggr]) remove_all_extents_t<T> shall be a complete type, an array type, or (possibly cv-qualified) void.	
US 144		20.17.5 [time,duration]		Te	Add a deduction guide for class template duration	Add to <chrono> synopsis: template <class Rep, class Period> duration(const Rep &) -> duration<Rep>;	
US 145		21.3.1 [basic.string]		Te	There is no requirement that traits::char_type is charT, although there is a requirement that allocator::value_type is charT. This means that it might be difficult to honour both methods returning reference (such as operator[]) and charT& (like front/back) when traits has a surprising char_type. It seems that the allocator should NOT rebind in such cases, making the reference-returning signatures the problematic ones.	Add a requirement that is_same_v<typename traits::char_type, charT> is true, and simplify so that value_type is just an alias for charT.	
US 146		23.2.1 [container.requirements.general]	13	Te	An allocator-aware contiguous container must require an allocator whose pointer type is a contiguous iterator. Otherwise, functions like data for basic_string and vector do not work correctly, along with many other expectations of the contiguous guarantee.	Add a second sentence to **23.2.1 [container.requirements.general]** p13: An allocator-aware contiguous container requires allocator_traits<Allocator>::pointer is a contiguous iterator.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 147		23 [containers]		Te	One of the motivating features behind deduction guides was constructing containers from a pair of iterators, yet the standard library does not provide any such deduction guides. They should be provided in header synopsis for each container in clause 23. It is expected that the default arguments from the called constructors will provide the context to deduce any remaining class template arguments, such as the Allocator type, and default comparators/hashers for (unordered) associative containers. At this stage, we do not recommend adding additional guides to deduce a (rebound) allocator, comparator etc. due to the likely large number of such guides. It is noted that the requirements on iterator_traits to be an empty type will produce a SFINAE condition to allow correct deduction for vector in the case of the Do-The-Right-Thing clause, resolving ambiguity between two integers, and two iterators.	For each container in clause 23, add to the header synopsis a deduction guide of the form: template <class Iterator> container(Iterator, Iterator) -> container<typename iterator_traits<Iterator>::value_type>;	
US 148		23.3.2 [array.syn]		Te	std::array does not support class-template deduction from initializers without a deduction guide.	Add to <array> synopsis: template <class TYPES> array(TYPES&&...) -> array<common_type_t<TYPES...>, sizeof...(TYPES)>;	
US 149		23.3.7.3 [array.specaial]	3	Ed	The array swap function also exchanges the values of elements, which is forbidden (unless explicitly documented) by **23.2.1 [container.requirements.general]** p9	Update the note accordingly.	
US 150		23.6		Te	The three container adapters should each have a	For each container adapter, add a deduction guide	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
		[container.adaptors]			deduction guide allowing the deduction of the value type T from the supplied container, potentially constrained to avoid confusion with deduction from a copy/move constructor.	of the form: template <class Container> adapter(const Container&) -> adapter<typename Container::value_type, Container>;	
US 151		24.5.2 [insert.iterators]		Te	The three insert iterators should each have an instantiation guide to initialize from a container.	Add to the <iterator> header synopsis: template <class Container> back_insert_iterator(Container&) -> back_insert_iterator<Container>; template <class Container> front_insert_iterator(Container&) -> back_insert_iterator<Container>; template <class Container> insert_iterator(Container&, typename Container::iterator) -> insert_iterator<Container>;	
US 152		24.6.1.1 [istream.iterator.cons]		Ed	*see below* for the default constructor should simply be spelled constexpr. The current declaration looks like a member function, not a constructor, and the constexpr keyword implicitly does not apply unless the instantiation could make it so, under the guarantees already present in the Effects clause.	Replace *see below* with constexpr in the declaration of the default constructor for istream_iterator in the class definition, and function specification.	
US 153		24.6.1.1 [istream.iterator.cons]		Te	istream_iterator default constructor requires a DefaultConstructible T	Add a new p1: *Requires:* T is DefaultConstructible	
US 154		24.6.1.1 [istream.iterator.cons]	5	Te	The conflation of trivial copy constructor and literal type is awkward. Not all literal types have trivial copy constructors, and not all types with trivial copy constructors are literal.	Revise p5 as: Effects: Constructs a copy of x. If T has a trivial copy constructor, then this constructor shall be a trivial copy constructor. If T has a constexpr copy	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
						constructor, then this constructor shall be constexpr.	
US 155		24.6.1.1 [istream.iterator.cons]	7	Te	The requirement that the destructor is trivial if T is a literal type should be generalized to any type T with a trivial destructor - this encompasses all literal types, as they are required to have a trivial destructor.	Revise p7 as: *Effects:* The iterator is destroyed. If T has a trivial destructor, then this destructor shall be a trivial destructor.	
US 156		25 [algorithm], 26.8 [numeric.ops]		Te	Parallel algorithms cannot easily work with InputIterators, as any attempt to partition the work is going to invalidate iterators used by other sub-tasks. While this may work for the sequential execution policy, the goal of that policy is to transparently switch between serial and parallel execution of code without changing semantics, so there should not be a special case extension for this policy. There is a corresponding concern for writing through OutputIterators. Note that the input iterator problem could be mitigated, to some extent, by serially copying/moving data out of the input range and into temporary storage with a more favourable iterator category, and then the work of the algorithm can be parallelized. If this is the design intent, a note to confirm that in the standard would avoid future issues filed in this area. However, the requirement of an algorithm that must copy/move values into intermediate storage may not be the same as those acting immediately on a dereferenced input iterator, and further issues would be likely. It is not clear that anything can be done to improve the serial nature of writing to a simple output iterator though.	All algorithms in the <algorithm> and <numeric> headers that take an execution policy and an InputIterator type should update that iterator to a ForwardIterator, and similarly all such overloads taking an OutputIterator should update that iterator to a ForwardIterator. (Conversely, if the design intent is confirmed to support input and output iterators, add a note to state that clearly and avoid confusion and more issues by future generations of library implementers.)	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 157		25 [algorithm], 26.8 [numeric.ops]		Ed	Many algorithms list parallel overloads in the header synopsis, but are not repeated under the specification sub-clause for the corresponding (serial) algorithm, unless they make substantive tweaks to the contract. This is confusing when looking up the specification for a given algorithm; the parallel overloads should be added directly under the serial forms without further change.	Ensure all parallel algorithm signatures appear above their corresponding specification, even when no change of contract from the serial form is intended.	
US 158		26.8 [numeric.ops]		Ed	The numerical algorithms in the <numeric> header have more in common with the algorithms library (clause 25) than they do with anything else in the numerics library (clause 26). In particular, there is front-matter on definitions that apply only to clause 25, that is later opted-into just the numeric-algorithms clause 26.8 [numeric.ops], and this became more pronounced with the addition of the parallel algorithm overloads. A more ambitious step would be to move the contents of the <numeric> header into <algorithm>, retaining it as a deprecated header whose contents are the single line #include <algorithm>. That discussion is probably better deferred to the next revision of the standard though.	Move **26.8 [numeric.ops]** into clause 25, preceding **25.6 [alg.c.library]**. Move **26.2 [numeric.defns]** under **25.1 [algorithms.general]**. Move **20.9 [execpol]** into clause 25, somewhere before the specification of the <algorithm> header.	
US 159		26.8.3 [Reduce]		Te	GENERALIZED_SUM should be available for only parallel versions of the algorithm. Permuting the operands should not be permitted for non-parallel versions, in which case reduce is equivalent to accumulate.	*Returns:* GENERALIZED_NONCOMMUTATIVE_SUM(...). Repeat exactly the current contract for the overloads with a parallel policy (including the serial policy).	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 160		26.8.4 [transform.reduce]		Te	transform_reduce(begin(vector_strings), end(vector_strings), upcase, "", concat) should not reorder the strings. The serial form of this algorithm (i.e., with no execution policy; no change for the explicit serial policy) should return a GENERALIZED_NONCOMMUTATIVE_SUM rather than the specified GENERALIZED_SUM.	*Returns:* GENERALIZED_NONCOMMUTATIVE_SUM(...). Repeat exactly the current contract for the overloads with a parallel policy (including the serial policy).	

US 161		26.8.5 [inner.product]		Te	There is a surprising sequential operation applying BinaryOp1 in inner_product that may, for example, require additional storage for the parallel algorithms to enable effective distribution of work, and is likely to be a performance bottleneck. GENERALIZED_SUM is probably intended here for the parallel version of the algorithm, with the corresponding strengthening on constraints on BinaryOp1 to allow arbitrary order of evaluation.	For the overloads taking an execution policy, copy the current specification, but replace algorithm in Effects with:
						
						GENERALIZED_SUM(plus<>(), init, multiplies<>(*i1, *i2), ...)
						
						GENERALIZED_SUM(binary_op1, init, binary_op2(*i1, *i2), ...)	

US 162		26.8.11 [adjacent.difference]		Te	The specification for adjacent_difference has baked-in sequential semantics, in order to support reading/writing through input/output iterators. There should a second specification more amenable to parallelization for the overloads taking an execution policy.	Provide a specification for the overloads taking an execution policy this is more clearly suitable for parallel execution. (i.e., one that does not refer to an accumulated state.)	
US 163		30.6.3 [futures.future_error]		Te	The constructor for future_error should not be exposition only - this is the only exception class in the standard library that users have no clearly specified way to throw themselves. If we want the exception class to be limited to the standard library, at least make the exposition-only constructor private.	Document the exposition-only constructor.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 164		30.6.7 [futures.shared_future]		Te	Add a deduction guide for creating a shared future from a future rvalue.	Add to the <future> synopsis: template <class R> shared_future(future<R>&&) -> shared_future<R>;	
US 165		30.6.9 [futures.task]		Te	The constructor that type-erases an allocator has all of the problems of the similar function constructor that was removed for this CD. This constructor from 'packaged_task' should similarly be removed as well. If we prefer to keep this constructor, the current wording is underspecified, as the Allocator argument is not required to be type satisfying the Allocator requirements, nor is allocator_traits used.	Strike ~~template <class F, class Allocator>~~ ~~packaged_task(allocator_arg_t, const Allocator& a, F&& f);~~ from the class definition in p2, and from **30.6.9.1 [futures.task.members]** p2. Strike the last sentence of 30.6.9.1p4. In p3, revise "These constructors" to "This constructor"	
US 166		C.1 [diff.iso]		Ge	The C standard has lower limits for many implementation quantities, such as an #include recursion depth of 15 rather than 256 in C++. Suggest adding a compatibility clause for Annex B that observes that C often has lower implementation limits than C++, when trying to write portable code (without calling each out specifically, as that would be a maintenance burden for future standards).	Add **C.11 [diff.implimits]** with a paragraph that portable code intended to translate in both languages should be aware that C has lower implementation limits than C++. Strike **26.8.1 [numeric.ops.overview]** p1.	
US 167		25.2.4	2	te	Calling 'std::terminate' when an element access function exits via. an uncaught exception effectively disables the normal means of C++ error handling and propagation when using the parallel algorithms. This will be both confusing to users and a common source of bugs. Furthermore, by defining this behavior we are essentially preventing further solutions to this problem.	There are several solutions that would be acceptable, among them: 1. Make it undefined behavior when an element access function exits via. an uncaught exception. This will allow for a future solution to this problem that is backwards compatible. 2. When an element access function exits via. an uncaught exception, throw a 'std::exception_list'	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
						which represents a collection of exceptions that were thrown in parallel. 3. When an element access function exits via. an uncaught exception, throw an unspecified 'std::exception'. 4. Rename the parallel algorithms to clarify that exception throwing code will result in a call to 'std::terminate'. For example 'std::exceution::parallel_policy' would be renamed to 'std::exceution::parallel_policy_noexcept' and 'std::execution::par' would be renamed to 'std::execution::par_noexcept'.	
US168		25.2.5	2	te	It is unclear what behavior a parallel algorithm will have when a user-provided function exits via. an uncaught exception. This statement seems to require most parallel algorithms to nodeterministically choose one of the exceptions thrown and then re-throw that in the calling thread.	Clarify in section 25.2.5 what happens when a user-provided function throws an exception.	
US 169		25.2.5	2	te	This statement seems to require most parallel algorithms to nodeterministically choose one of the exceptions thrown and then rethrow that in the calling thread. In the case that multiple threads witness an exception from a user-provided function, all but one of those exceptions gets discarded. It is much preferrable to have all exception data preserved.	When a user-provided function exits via. an uncaught exception, throw a 'std::exception_list' structure which represents a collection of exceptions that were thrown in parallel.	
US 170	2	25.2.4		te	The current wording does not leave the door open for executors (a feature under development by SG1) to modify the exception-handling behaviour of parallel algorithms in the future without breaking backwards compatibility.	Define a construct std::execution::exception_handling (the “parallel algorithms exception handling customization point”) such that std::execution::exception_handling(ep), where ep is an ExecutionPolicy, is well formed and returns an object which fulfils a ParallelExceptionHandler concept. For the three execution policies defined in the standard, std::execution::exception_handling(ep) shall return a parallel exception handler object which shall call	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
						terminate() when the invocation of an element access function exits via an uncaught exception. The intention of this wording is to cause **no change** to the behaviour in the existing wording, but to ensure that the “terminate() on uncaught exception” behaviour is not baked into all future executors, just the implicit “default executor”.	
US 171		20.15.2		te	The *_constant<> templates (including the proposed addition, bool_constant<>) do not make use of the new template<auto> feature.	Add a constant<> (subject to bikeshedding) template which uses template<auto>. Define integral_constant<> as using integral_constant<T, V> = constant<T(V)> or integral_constant<T, V> = constant<V>. Either remove bool_constant, define it as using bool_constant = constant<bool(B)> or using bool_constant = constant<B>.	
US 172		17.7, 26.9 and possibly others		ge	noexcept is inconsistently applied across headers which import components of the C standard library into the C++ library; some functions (std::abort(), std::_Exit(), etc) are defined as noexcept in some places, but not in others. Some functions which seem like they should be noexcept (std::abs(), std::div(), etc) are not defined as noexcept.	Make the majority of the C library functions (with exceptions such as std::qsort() and std::bsearch(), which can call user code) noexcept. The following comments address areas of particular concern.	
US 173		17.7		ed	In the header synopsis for <cstdlib>, std::abort(), std::atexit() (both overloads), std::at_quick_exit() (both overloads), std::_Exit() and std::quick_exit() are **not** declared noexcept. However, in 18.5 they are declared noexcept.	Add noexcept to the declarations of std::abort(), std::atexit(), std::at_quick_exit(), std::_Exit() and std::quick_exit() in 17.7.	
US 174		17.7 and 18.5		te	std::exit() is not noexcept.	Make std::exit() noexcept.	
US 175		26.9 and 26.9.2		te	std::abs(), std::labs() and std::llabs() are not noexcept.	Make all overloads of std::abs(), std::labs() and std::llabs() noexcept.	
US 176		17.7		te	std::div(), std::ldiv() and std::lldiv() are not noexcept.	Make all overloads of std::div(), std::ldiv() and std::lldiv() noexcept.	
US177		26.9		te	None of the functions in namespace std in <cmath> are noexcept.	Make all of the functions in namespace std in <cmath>, including the new special math functions, noexcept.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
US 178		20.10.11		te	The C library memory allocation functions declared in <cstdlib> (std::aligned_alloc(), std::calloc(), std::malloc(), std::realloc() and std::free()) are not noexcept.	Make std::aligned_alloc(), std::calloc(), std::malloc(), realloc() and std::free() noexcept.	
US 179		20.6.3		ed	The heading for this section is “optional for object types”, yet there are no specializations (partial or otherwise) of this optional class or other optional classes defined in the standard.	Change the heading to “Class optional”. Change the stable tag to optional.class (following the style of any.class, etc).	
US 180		20.7.2		ed	The heading for this section is “variant of value types”, yet there are no specializations (partial or otherwise) of this variant class or other variant classes defined in the standard.	Change the heading to “Class variant”. Change the stable tag to variant.class (following the style of any.class, etc).	
US 181	1	20.7.2		te	Support for void alternatives in variant is inconsistent. Incomplete types are normally disallowed in variant. 20.7.2.1 states that “When an instance of variant holds a value of alternate type T, it means that a value of type T [snip] is allocated within the storage of the variant object”; this implies that variant requires its alternatives of object type to be complete types (the size of which can be determined). Thus, it is illformed to try to construct a variant<monostate, Incomplete> v (where Incomplete is an incomplete type) because we cannot determine the size needed to store Incomplete. However, variant allows (possibly cv-qualified) void as an alternative type. Since void can never be completed (3.9.1) it seems that variant just assumes it has a size of 0 and requires no storage. However, you cannot copy, move or swap a variant with an alternative of void type.	• Disallow void alternative types as they are incomplete or • Rely on the fact that void alternatives take no part of the embedded storage and ignore them when a complete type would otherwise be required.	
US 182		26.8.5		ed	One of the types given in the signature of inner_product() is “Input**g**terator” [sic].	s/Inputgterator/InputIterator/	
US 183		25.1 and 26.8.1		ge	The current wording of the standard makes it very tricky to determine whether an algorithm has a parallel (e.g. ExecutionPolicy) overload. The header synopses for <algorithm> and <numeric> list the ExecutionPolicy overloads, but the definitions do not list the overloads (which can be understood by	• Add ExecutionPolicy overloads to all the relevant definitions, or • Add a note in the definition of all algorithms which **do not** have ExecutionPolicy overloads stating that they have no such	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. US for United States; 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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					reading 25.2.5.2, which essentially states that unless noted otherwise, the ExecutionPolicy overloads have the same semantics and are thus not listed in the definitions). This makes it hard to determine whether an algorithm has an ExecutionPolicy overload. For example, 25.3.1, which defines all_of(), does not list an ExecutionPolicy overload, but all_of() **does** have such an overload. On the other hand, 25.5.6.1, which defines push_heap(), also does not list an ExecutionPolicy overload, and push_heap() **does not** actually have such an overload.	overload (e.g. accumulate(), push_heap). • Add a table listing all the algorithms in <numeric> and <algorithm> which **do** have ExecutionPolicy overloads, or • Add a table listing all the algorithms in <numeric> and <algorithm> which **do not** have ExecutionPolicy overloads.	
US 184		26.8.1		te	An ExecutionPolicy overload for inner_product() is specified in the synopsis of <numeric>. Such an overload seems impractical. inner_product() is ordered and cannot be parallelized; this was the motivation for the introduction of transform_reduce().	Delete the ExecutionPolicy overload for inner_product().	
US 185		27.10.7		te	The filesystems library provides two function signatures for (most, possibly all) of the free functions in its interface; one signature which takes a reference to an error_code (reporting errors by assigning to the reference and returning) and one which does not (reporting errors by throwing an exception). In addition to adding a large number of overloads, this approach makes it very tedious for programmers to write generic functions which use the filesystem library. If the author of such a function wishes to provide both error_code and exception-throwing interfaces (in the same way the filesystem library does), two different versions of the generic function must be written. This may also be a burden to implementers.	Define a global error_code object called std::throws, and change all the function signatures in the filesystem library to have the form R f(/*…*/, error_code& ec = throws). If an error occurs in the function, if ec is the same object as throws (&ec = &throws), then an exception is thrown. Otherwise, an error code is created and assigned to the reference ec. This should **not** change the interface or error handling behaviour of the filesystem library. This approach has been used in the HPX library and (IIRC) the Boost libraries including Boost.Filesystem..	
End							

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

1Template for comments and secretariat observations

Date: 2016-09-16	Document: **SC22 WG21 N4604**	Project: CD 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
GB 1		1.1	p2	Te	Paper P0063R3 changed our normative reference to C to refer to C11 not C99, but missed one important reference: in [intro.scope](1.1) paragraph 2, where we define the term "C standard", we still define it as referring to C99 rather than C11.	It seems correct to also update that reference to refer to C11, *except* that we will need corresponding updates to [diff.iso] (Annex C.1) to describe the C11 language features not available in C++.	
GB 2		1.2	(1.1)	Te	The latest ECMAScript standard was released in June 2016, while the current CD references the 1999 Third Edition. ECMAScript is used only to define the default grammar for regular expressions.	Update the reference in (1.1) to ECMA-262 ECMAScript 7th Edition/June 2016, or to the last revision adopted by ISO, ISO 16262:2011. Update the section reference in "Table 127 - regex_constants::match_flag_type effects…" for format_default Review [re.grammar]	
GB 3		1.2	(1.5)	Te	Latest POSIX standard is ISO/IEC 9945:2009/Cor 1:2013, rather than the 2003 standard referenced here. The current document uses POSIX to define some error constants, define filesystem operations, and define several regular expression grammars.	Update the POSIX reference to ISO/IEC 9945:2009/Cor 1:2013. Consider any updates to [cerrno.syn], the errc enumerators in [system_error.syn] and additional concerns for [filesystems]	
GB 4		1.2	(1.6)	Te	ISO standards are only supposed to have normative references to the latest version of other ISO standards, yet the C++17 CD still refers to ISO/IEC 10646-1:1993, Information technology — Universal Multiple-Octet Coded Character Set (UCS)— Part 1: Architecture and Basic Multilingual Plane.	Update 1.2 [intro.refs] to the current 10646 standard and make any necessary subsequent changes to wording.	
GB 5		1.3.17		Ge	The definition of the term template parameter should be more than naming a single grammar term, to help distinguish it from all the other definitions of 'parameter' that include a plain-english description	Enhance the definition of 'parameter' with a plain English description of a template parameter.	
GB 6		1.3.25		Ge	The definition of undefined behavior does not allow for the requirement that 'constexpr' functions are required to diagnose undefined behavior in constant evaluation contexts. This also affects what we say for SFINAE: you get a	Add the extra requirement for constexpr	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					substitution failure if the substituted type *would be* ill-formed (but you don't actually form it in that case, so the program is not ill-formed); you get a non-constant expression if the evaluation *would have* undefined behaviour (but you don't actually evaluate it in that case, so the behaviour is not undefined).		

GB 7		1.8	(3.3)	Ed	The 3rd bullet is confusing, as it is not clear where a smaller array would come from	Provide an example of where a smaller array would come from:
						
						`struct A {`
						`  unsigned char a[32];`
						`};`
						`struct B {`
						`  unsigned char b[16];`
						`};`
						`A a;`
						`B *b = new (a.a + 8) B;`
						`int *p = new (b->b + 4) int;`
						
						Here, two array objects satisfy the first two bullets for the int object denoted by *p, namely `a.a` and `b->b`. The third bullet says that `b->b` provides storage for the `int` but `a.a` does not.	

GB 8		1.8	5	Ed	The definition of 'complete object' is confusing: "If x is a complete object, then x is the complete object of x. Otherwise" … with the inference that if otherwise is not triggered, the former must have been true.	Clarify the two uses of complete object in the sentence, perhaps "If x is a complete object, then the complete object of x is itself."	
GB 9		1.8	7	Te	base class objects of zero size is a misleading term, as 'sizeof' such an object is non-zero. Size should not be a property of an object, rather than	A better statement is that 'empty' base class objects can share the address of a non-empty sub-object, so reword to talk about	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					a type.	base class sub-objects sharing storage, rather than having zero size.	
GB 10		1.11		Ge	ECMAScript is a registered trademark of ECMA, and should be added to our list of acknowledgements.	Add a new paragraph: ECMAScript is a registered trademark of Ecma International.	
GB 11		1.7		Ed	While the number of bits in a byte is implementation-defined, it is also exposed directly in code as the CHAR_BIT macro in <limits.h> from the C library,and <climits> in the C++ library.	Add a footnote pertaining to "the number of which is implementation-defined" saying "The number of bits in a byte is reported by the macro CHAR_BIT in the header <climits>."	
GB 12				Ge	The BSI would like to ensure that outstanding issues on the issues lists are all considered before the final IS is produced.		

GB 13		5.2.3	p2	Te	The wording for template parameter deduction for constructors allows:
					
					`template-name foo(a,b,c);`
					`template-name foo{a,b,c};`
					`template-name(a,b,c)`
					
					… but not …
					
					`template-name{a,b,c}`
					
					(as the wording in 5.2.3p2 only covers the case of a template-name followed by a parenthesized expression-list)
						Add wording to 5.2.3p2 to allow the problematic case:
						
						A template-name corresponding to a class template followed by a parenthesized expression-list<ins> or by a braced-init-list</ins>...	

GB 14		5.3.2		Te	C++17 removed pre-incrementing on objects of type bool. However, the last sentence in 5.3.2 was not changed to reflect this: "If x is not of type bool, the expression ++x is equivalent to x+=1".	Change the last sentence in 5.3.2 to "The expression ++x is equivalent to x+=1."	
GB 15		5.1.5	18	Te	CWG 2011 fixes a regression from C++14, introduced by the resolution of CWG 2012. This regression causes many existing	Accept the proposed wording for CWG 2011 or similar wording that permits references captured by reference to be used outside	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					C++14 programs to have undefined behavior in C++17. Example: `auto f(int &r) { return [&]{++r;} } void g(int n) { f(n)(); }`	their lifetime.	
GB 16		7	8	Te	Decomposition declarations are allowed at namespace scope, so it should be possible to specify their linkage.	Allow static, extern, thread_local, and inline specifiers, or disallow decomposition declarations at namespace scope.	
GB 17		7	8	Te	Decomposition declarations only allow cv qualifiers and auto in the decl-specifier-seq. There seems to be no reason to disallow constexpr, and it would be useful to allow it.	Permit constexpr specifier.	

GB 18		8.5	1	Te	The rules for auto deduction and template argument deduction do not match the rules for decomposition declarations when the initializer is an array.
					
					`int some_array[3];`
					`auto [a, b, c] = some_array; // deduces int[3]`
					`auto x = some_array; // deduces int*`
					
					This prevents reliable refactoring of `auto [a, b, c] = e;` into `auto x = e; auto &[a, b, c] = x;` and makes the rules for `auto` deduction unnecessarily complex.
						Remove the special case for copying arrays by value in decomposition declarations.	

GB 19		8.6.3	5	Te	This code used to be valid and is now ill-	When a temporary object is materialized so a reference to *cv* T can bind to it, the created	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

					formed:
					
					`const int &r = 1;`
					`constexpr int n = r;`
					
					because p0135's changes to [dcl.init.ref] don't provide proper cv-qualification for the created temporary object.
						temporary object should be qualified by *cv*.	

GB 20		8.5	3	Te	If the user specializes tuple_size for their type, but messes up the definition of value somehow:
					
					`  template<> struct std::tuple_size<MyPair> {`
					`    const int value = 2;`
					`  };`
					
					we will silently fall back to memberwise decomposition. This is user-hostile.
						Commit to the tuple-like interpretation if `tuple_size`<E> is a complete type.
						Change 8.5/3 to:
						
						"Otherwise, if the qualified-id ::std::tuple_size<E> names a complete type, the expression ::std::tuple_size<E>::value shall be a well-formed integral constant expression and the number of elements in the identifier-list shall be equal to its value. […]"	

GB 21		13.3.1.8	1.1	Te	The addition of implicit deduction guides causes class template argument deduction to silently do the wrong thing in many cases, including some in the standard library. Fixing a bad deduction in a later version of a library is a breaking change if anyone is using the bad deduction. For example, with the current standard wording, `std::tuple(a, b, c)` and `std::make_tuple(a, b, c)` will do *different* things in some cases.	Delete bullet 1 of 13.3.1.8/1, removing implicit deduction guides from constructors of the primary template.	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					Once we ship this, we would not be able to change `std::tuple(a, b, c)` to match `make_tuple` without risk of breaking existing code.		
GB 22		15	3		This sentence twice refers to "exceptions raised while destroying" objects, but the term is not defined - exceptions are thrown, not raised. This also affects Table 29 - Allocator Requirements on the 'a.allocate. row, and a Note in 30.3.1.3p1 [thread.thread.destr].	Change all uses of 'raise' and 'raised', where they apply to exceptions, to 'throw' and 'thrown'.	

GB 23		15.3	2	Te	As functions and arrays decay to pointers when thrown, it is not possible to catch such a type by reference. This is partially acknowledged by the implicit function/array-to-pointer decay that occurs in a handler. Ideally it should be ill-formed to write such a handler, to avoid unusual mistakes; otherwise, it would merit a note that such nonsensical handlers are allowed for code like:
					
					`template <typename T>`
					`void test() {`
					
					`try {`
					
					`T t = {};`
					`throw t;`
					
					`}`
					`catch(T const &) {`
					`}`
						Add a note with the example from this comment.	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					`}` `test<int[8]>();` *will not catch the 'int *' exception*		
GB 24		15.3	4	Ed	The given example for a handler that cannot be entered is invalid, as a handler for a derived class can still be activated after the handler for an ambiguous base.	Add 'final' and 'unambiguous public' to the example: "for example by placing a handler for a <ins>final</ins> derived class after a handler for a corresponding <ins>unambiguous public</ins> base class."	
GB 25		15.1	7	Te	If an exception is rethrown, it might also want to call terminate for a function exiting by an exception. Destructors are already covered by separate wording, but I believe a copy-constructor in a handler that catches by value relies on this clause to trigger the 'terminate' call. However, this highlights a problem with the current wording when such a copy constructor throws and catches an exception by calling a function that throws from within the constructor's compound statement.	Add wording to cover the additional case.	
GB 26		15.1	4	Te	Which active handler is the 'last' when two threads are handling the same exception object? Is there some implicit sequencing relation between handlers in different threads? A potential data race, if both threads think they are 'last' and destroy the same object? A potential leak as neither thinks it is 'last'? There is also a question of whether `exception_ptr` destructors for the same exception object synchronize with		

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					each other (even in the case where the count does not drop to 0).		
GB 27		15.5.3		Te	exception_ptr and rethrow_exception allow the same exception object to be active multiple times in the same thread. It is not clear if 'uncaught_exceptions' should count such cases as a single exception object, or should count each activation of the same object in the current thread.		
GB 28		17		Te	The C++ standard library provides many \`constexpr\` global variables. These all create the risk of ODR violations for innocent user code. This is especially bad for the new \`ExecutionPolicy\` algorithms, since their constants are always passed by reference, so any use of those algorithms from an inline function results in an ODR violation. This can be avoided by marking the globals as \`inline\`.	Add `inline` specifier to: — `bind` placeholders `_1`, `_2`, `...` — `nullopt`, `piecewise_construct`, `allocator_arg`, `ignore` — `seq`, `par`, `par_unseq` in `<execution>`	
GB 29		17.3.2 17.3.26		Ed	The definition of blocking is part of the execution model defined in 1.9, so this definition should move to clause 1, which covers the whole standard and not just the library.	Move subclauses [defns.block] and [defns.unblock] under section 1.3 [intro.defs].	
GB 30		17.3.17		Te	The definition of 'object state' applies only to class types, implying that fundamental types and arrays do not have this property.	Replacing "an object state" with "a value of an object" in 17.3.27 and dropping the definition of "object state" in 17.3.17	
GB 31		17.3.25		Ed	The term character traits appears to be defined in a non-normative note.	Provide a distinct clause to define the term character traits, change the term to non-italic so it does not appear to be a definition, or add a cross-reference if it is calling out a specific existing definition of the term.	
GB 32		17.4		Ed	This subclause does not deserve a separate title, number, and stable-name. It would	Move 17.4 [defns.additional] p1 as a [Note:], forming the new p1 of 17.3 [definitions], and	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					serve better as a [Note:] at the top of the preceding clause, which provide the definition of terms for the library.	remove the corresponding title and stable name.	
GB 33		17.5.2.3	3	Ed	Is 'external behavior' a well-defined term, or is 'observable behavior' the intent?	Replace 'external behavior' with 'observable behavior'.	
GB 34		17.6.1.1	1	Ed	Macros are not entities, see 3p3 [basic] for the definition. A better way to say this should be found, or perhaps a footnote against the macro term, to grandfather the casual library usage here.	There's another (different) list of what's in the library in 1.5p2 ("templates, classes, functions, constants, and macros"). Neither list seems complete. Perhaps we could use "entities and macros" in both 1.5p2 and 17.6.1, strike 17.6.1.1p1, and then strike "macros" from 17.6.1.1p2?	
GB 35		17.6.5		Te	Most implementations have poor testing and support for instantiating standard library templates with volatile-qualified types. We should grant a library-freedom to conforming implementations so that support for volatile (and const volatile) qualified types in standard library templates is not required unless explicitly specified - and mandate such support for all templates in the <type_traits> header. Additional support is already specified in most places we would be interested (e.g., tuple API). We may want to additionally guarantee support through forwarding references.	add a new 17.6.5.x Volatile Qualified Types [res.on.volatile.type] describing the intended level of support for volatile qualifiers.	
GB 36		17.6.5.11	(3.2)	Te	For bullet (3.2), no base classes are described as non-virtual. Rather, base classes are not specified as virtual, a subtly different negative.	Rewrite bullet 3.2: Every base class not specified as virtual shall not be virtual;	
GB 37		17.7		Ed	The whole structure of the library clauses, explicitly documented in 17.1 [library.general], precluded specifying library headers in clause 17. This C header should	Move this to clause 18	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					be documented either in clause 18, clause 20, or split between the two, with the parts mandatory for a free-standing implementation at least appearing in clause 18.		
GB 38		17.6.5.6		Te	Relax the prohibition on libraries adding constexpr; this was a constraint requested by library implementers when constexpr was new, and those same implementers now feel unduly constrained.	Rewrite the whole sub-clause to support libraries adding constexpr in a compatible manner, much like the freedom to add a noexcept specification.	
GB 39		17.6.5.4	4	Ge	The example is supposed to highlight the 'otherwise specified' aspect of invoking ADL, yet there is no such specification. It is unlikely that we intend to explicitly qualify calls to operator functions, so they probably should be exempted from this restriction.	Fix example (and referenced clause) to specify use of ADL, or exempt operators from this clause, and find a better example, probably using swap.	
GB 40		17.6.5.12	Footnote 189	Ge	The freedom referenced in footnote 189 was curtailed in C++11 to allow only non-throwing specifications. The footnote is both wrong, and unnecessary.	Strike footnote 189	
GB 41		17.6.5.12	2,4	Te	The "any other function" sentence in p4 contradicts the restriction placed in p2.	Strike the third sentence of p4, starting with "Any other function…". Consolidate its implementation-defined requirements into p2, along with footnote 188.	
GB 42		17.6.5.12	Footnote 188	Ge	The word 'should' makes footnote 188 sound like normative encouragement, if not an actual mandate.	Either use a non-loaded word, such as "typically", or move footnote 188 directly into the main text.	
GB 43		17.6.5.12	1,4	Ed	The freedom to add exception specifications is repeated in p1 and p4, in slightly different terms, highlighting the dangers of redundancy in a specification.	Consolidate the two sentences into a new p5, as per p0003r5.	
GB 44		20		Te	P0067R3 was moved at Oulu but not applied to the working paper due to a major technical error discovered by the project editor (the signatures in the synopsis for	Apply the revised wording in P0067R4	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					`from_chars` did not match the detailed wording).		

GB 45		20		Te	If P0067R4 is applied consider how to parse hexadecimally:
					
					`to_chars(beg, end, 42, 16);` 16 for hex
					`to_chars(beg, end, 4.2, true);` `true` means hex
					`to_chars(beg, end, 4.2, chars_format::hex);`
					`to_chars(beg, end, 4.2, chars_format::hex, 2);`
					
					That is: We have 3 different formats to specify hex depending on value types and whether to use precision.
					Which application programmer should remember this?
					
					May be even worse (I am not sure):
					
					`to_chars(beg, end, 4.2, 16);`
					
					would silently convert 4.2 to 4 and
					
					`to_chars(beg, end, 4, chars_format::hex);`
					
					would silently convert 4 to 4.000000.
						The various options should be harmonized, possibly by use of an extended enum approach, having the values:
						
						dec, hex, scientific, fixed, general
						
						with dec (new!) as default for integral values and general for floats	

GB 46		20.2		Te	in_place_tag is an implementation detail that should not be exposed to the user.	The declaration should be marked as exposition-only to allow implementors to use a name in the implementation namespace (such as __in_place_tag) for the type.	
GB 47		20.11.2		Ed	The approval of P0220R1 should have	Apply the changes from P0414R1.	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					added shared_ptr<T[]> and shared_ptr<T[N]> support to C++17, but due to editorial conflicts the change didn't get applied to the WP.		
GB 48		20.19.7 [parallel.execpol.objects]		Ed	[parallel.execpol.objects] is a subclause of [execpol] and is adjacent to [execpol.par], [execpol.vec] etc. There is no reason for it to have the prefix "parallel".	Change name [parallel.execpol.objects] to [execpol.objects].	
GB 49		20.6.5 [optional.bad_optional.access]		Te	https://issues.isocpp.org/show_bug.cgi?id=72 suggests changing the base class of std::bad_optional_access, but the issue appears to have been forgotten.	Address LEWG issue 72, either changing it for C++17 or closing the issue.	
GB 50		20.17.5 [time.duration], 20.17.6 [time.point]		Te	The reference implementation in P0092R1 is non-conforming, because it uses `++t` in the body of `round(const duration<R,P>&)` and that member function is not constexpr. A conforming implementation must do `t = t + ToDuration?(1)` or `t = ToDuration?(t.count() + 1)`. The straightforward increment should work in constant expressions.	Make all the member functions of duration and time_point constexpr.	
GB 51		20.14.3 [func.invoke]		Te	The function template std::apply() in [tuple.apply] is required to be constexpr, but std::invoke() in [func.invoke] isn't. The most sensible implementation of apply_impl() is exactly equivalent to std::invoke(), so this requires implementations to have a constexpr version of invoke() for internal use, and the public API std::invoke, which must not be constexpr even though it is probably implemented in terms of the internal version.	Add 'constexpr' to std::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

Date: 2016-09-16	Document: **SC22 WG21 N4604**	Project: CD 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
GB 52		20		Ed	There are several new stable names that are unnecessarily long, (and use underscores which look quite ugly due to the formatting of stable names). For example [optional.bad_optional.access], which could be called [bad.optional.access] or [optional.bad.access] instead. As an example of a sensible name, see [time.point] which is not called [time.time_point] even though that would be the "obvious" choice. Other culprits are [memory.polymorphic.allocator.class], [memory.resource.monotonic.buffer.ctor], and [func.searchers.boyer_moore_horspool.creation] Most of these seem to be in Clause 20, but there are other examples in other Clauses.	Review stable names for new clauses added since C++14. Consider abbreviating them instead of using complete unabridged class names.	
GB 53		20.14.3 [func.invoke]		Te	std::invoke can be made trivially noexcept using the new std::is_nothrow_callable trait:	Add the exception specifier noexcept(is_nothrow_callable_v<F(Args&&…)>) to std:invoke	
GB 54		20.8.2 [any.bad_any_cast]		Te	There is no specification for bad_any_cast.what.	Add a paragraphs: const char* what() const noexcept override; Returns: An implementation-defined NTBS. Remarks: The message may be a null-terminated multibyte string (17.5.2.1.4.2),	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						suitable for conversion and display as a wstring (21.3, 22.4.1.4).	
GB 55		20.13.6		Te	It is becoming more and more apparent that using a function type as the template argument to result_of causes annoying problems. That was done because C++03 didn't have variadic templates, so it allowed an arbitrary number of types to be smuggled into the template via a single parameter, but it's a hack and unnecessary in C++ today. result_of<F(Args…)> has absolutely nothing to do with a function type that returns F, and the syntactic trickery using a function type has unfortunate consequences such as top-level cv qualifiers and arrays decaying (because those are the rules for function types). It might be too late to change result_of, but we should not repeat the same mistake for std::is_callable.	Possibly get rid of the `is_callable<Fn(ArgTypes?…), R>` specialization. Change the primary template `is_callable<class, class R = void>` to `is_callable<class Fn, class.. ArgTypes?>` and define a separate template such as `is_callable_r<class R, class Fn, class… ArgTypes?>` for the version that checks the return type. The resulting inconsistency might need to be resolved/improved upon.	

GB 56		20.5.2.6	4	Te	`#include <utility>`
					`struct X { int a, b; };`
					`const auto [x, y] = X();`
					
					results in a hard error, because it attempts to instantiate `std::tuple_size<const X>`, which is not SFINAE-friendly. If the `#include` or `const` is removed, the code works.
						One option is to resolve LWG issue 2770: make `std::tuple_size<const T>` SFINAE-friendly. Do not define a member named `value` if `std::tuple_size<T>::value` is not well-formed.
						
						Alternatively a core language change could be made.	

GB 57		22.5 [locale.stdcvt]		Ge	The contents of <codecvt> are underspecified, and will take a reasonable amount of work to identify and correct all of the issues. There appears to be a general feeling that this is not the best way to	Deprecate and move the whole of clause 22.5 [locale.stdcvt] to Annex D.	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					address unicode transcoding in the first place, and this library component should be retired to Annex D, along side <strstream>, until a suitable replacement is standardized		
GB 58		23.2.4 [associative.reqmts]	Table 86 - Associative Container Requirements	Te	P0083R3 adds new member functions which return 'insert_return_type', which has at least three members. It would be convenient to be able to use the type with a decomposition declaration: auto[ins, pos, node] = m.insert(std::move(n)); Because the precise number of members and their order is unspecified, and it isn't a pair or tuple, that isn't guaranteed to work. A custom return type was used because pairs and tuples do not have descriptive names for their members, but structured bindings make it convenient to give custom names to the members (although their order must still be known).	Consider adding overloads of tuple_size/get etc. that do the right thing for UniqueAssocContainer::insert_return_type structs, or returning a tuple, or returning a struct with named fields, instead.	
GB 59		24.6.3 [istreambuf.iterator]		Te	There is no specification for istreambuf_iterator::operator→. This operator appears to have been added for C++11 by LWG issue 659, which gave the signature, but also lacked specification.	Add specification	
GB 60		27.5.4.2 [fpos requirements]	Table 108	Ge	The requirements on the 'stateT' type used to instantiate class template 'fpos' are not clear, and the following Table 108 - Position type requirements is a bit of a mess. This is old wording, and should be cleaned up with better terminology from the Clause 17 Requirements. For example, 'stateT' might be require CopyConstructible?, CopyAssignable?, and Destructible. Several entries in the final column of the table appear to be post-conditions, but without the 'post' markup to clarify they are not	Clarify the requirements and the table	

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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					assertions or preconditions. They frequently refer to identifiers that do not apply to all entries in their corresponding 'Expression' column, leaving some expressions without a clearly defined semantic. If 'stateT' is a trivial type, is 'fpos' also a trivial type, or is a default constructor not required/supported?		
GB 61		30.4.2.1 [thread.lock.guard]		Te	P0156R0 changed std::lock_guard<T> to std::lock_guard<T…> This is an ABI break, because the mangled name of the type changes. lock_guard is not movable, so is unlikely to appear in function signatures, but the change would break binary compatibility for any API which took a lock_guard by reference (e.g. where a function must only be called while a lock is held, and the lock is passed in as "evidence" of the lock). Whether the benefit of the change is worth an ABI change should be considered.	Revert the changes from P0156R0. A separate type could be added for the variadic case.	
GB 62		30.6.7 [futures.shared_future]	3	Te	There is an implicit precondition on most shared_future operations that 'valid() == true', 30.6.7p3. The list of exempted functions seems copied directly from class 'future', and would also include copy operations for shared_futures, which are copyable. Similarly, this would be a wide contract that cannot throw, so those members would be marked noexcept.	Revise p3: "The effect of calling any member function other than the move constructor, the copy constructor, the destructor, the move-assignment operator, the copy-assignment operator, or valid() on a shared_future object for which valid() == false is undefined." … Add noexcept specification to the copy	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						constructor and copy-assignment operator, in the class definition and where those members are specified.	
GB 63		Annex B		Ge	What is recommended limit for number of captures in a lambda expression? Suggest using the same number as number of arguments to a function call, but could alternatively be the number of members allowed in a class.	Add to Annex B: Lambda-captures in one lambda expression [256].	
GB 64		Annex B		Ge	what is recommended limit for number of comma-separated expressions in an initializer list?	Add to Annex B: Initializer-clauses in a braced-init-list [1024].	
GB 65		Annex B		Ge	How many variables can be defined in a decomposition declaration? Should this be similar to the identifier-list limit for macros, at 255, or closer to the number of local variables that can be declared in a function, 1024?	Add to Annex B: Variables defined by a single decomposition declaration [256].	
GB 66		Annex C [diff.cpp11.basic]		Ed	[diff.cpp11.basic] in Annex C makes no mention of needing to replace sized delete if you replace non-sized delete, otherwise you get undefined behaviour.	Document the change from C++11.	
GB 67		Annex E		Ed	Annex E (normative) Universal character names for identifier characters [charname] This Annex is only referenced in the standard in one place - 2.10 [lex.name]. As such, it adds little value as an Annex.	Move the contents of Annex E into 2.10 [lex.name]	
GB 68		3.9 [basic.types]		Te	The term 'literal type' is dangerous and misleading, as text using this term really wants to require that a constexpr constructor/initialization is called with a constant expression, but does not actually tie the selected constructor to the type being 'literal'.	Verify the uses of the term in the Core and Library specifications and replace with something more precise where appropriate.	

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

1Template for comments and secretariat observations

Date: 2016-09-16	Document: **SC22 WG21 N4604**	Project: CD 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
GB 69		20.7.11 [variant.hash]	p1	Ge	The paragraph is really trying to say two different things, and should be split into two paragraphs, using standard terminology.	The first sentence should become a Requires: clause, as it dictates requirements to callers. The second sentence should be a Remarks: clause, at is a normative requirement on the implementation.	

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

Comments and secretariat observations

Date: 7-Oct-2016	Document: **SC22 N4604, ISO/IEC CD 14882**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

RU 1	8.6 [dcl.init]	paragraph 7	te	Make empty or fully-initialized const objects default initializable. From the user's point of view all the following structures have their variables initialized, so the behaviour must be consistent:
				struct A0 {};
				const A0 a0; // currently ill-formed
				
				struct A1 {
				    A1(){}
				};
				const A1 a1;
				
				struct A2 {
				    int i;
				    A2(): i(1) {}
				};
				const A2 a2;
				
				struct A3 {
				    int i = 1;
				};
				const A3 a3; // currently ill-formed
				
				This issue was reported as the DR 253 http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html#253.
					If a program calls for the default-initialization of an object of a const-qualified type T, T shall be a class type with either a constructor that initializes all subobjects or a user-provided default constructor.	

RU 2	20.15.2 [meta.type.synop]	paragraph 2	te	Failed prerequirement for the type trait must result in ill-formed program. Otherwise hard detectable errors will happen:	Add to the end of the [meta.type.synop] section: Program is ill-formed if precondition for the type trait is violated.	

1 **MB** = Member body (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
NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				#include <type_traits>
				
				struct foo;
				
				void damage_type_trait() {
				    // must be ill-formed
				    std::is_constructible<foo, foo>::value;
				}
				
				struct foo{};
				
				int main() {
				    static_assert(
				        // produces invalid result
				        std::is_constructible<foo, foo>::value,
				        "foo must be constructible from foo"
				    );
				}		

RU 3	23.3.7.1 [array.overview]	paragraph 3	te	Force the literal type requirement for the iterator and const_iterator in the std::array so that iterators of std::array could be used in constexpr functions.	Add to the end of the [array.overview] section: iterator and const_iterator shall be literal types.	
RU 4	21.2.3.1 [char.traits.specializations.char] 21.2.3.2 [char.traits.specializations.char16_t] 21.2.3.3		te	It is confusing to see a class that is marked with constexpr but is not usable at compile time. std::string_view uses std::char_traits in many constexpr methods and functions. Many std::char_traits functions are not constexpr. At least std::char_traits::find, std::char_traits::length and std::char_traits::compare functions must be marked with constexpr.	As proposed in P0426R0, add constexpr for functions std::char_traits::find, std::char_traits::length and std::char_traits::compare in all the 21.2.3.* [char.traits.specializations.*] sections: static constexpr int compare(const char_type* s1, const char_type* s2, size_t n); static constexpr size_t length(const char_type* s);	

1 **MB** = Member body (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
NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com- ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
	[char.traits.specializations.char32_t] 21.2.3.4 [char.traits.specializations.wchar.t]				static constexpr const char_type* find(const char_type* s, size_t n, const char_type& a);	
RU 5	all	all	ge	Writing comparisons for user defined classes is error prone and requires a lot of trivial typing, so it must be done by compiler when possible.	Fix that by continuing the work on "P0221R2: Proposed wording for default comparisons" or at least by accepting proposals that use user defined operator< and operator == to generate the remaining comparison operators.	
RU 6	all	all	ge	The adoption of the "constexpr if-statements" changes from document P0292R2 is a step in the right direction for code simplification.	Preserve the functionality and think of extending it in the future (for-constexpr statements, switch-constexpr statements).	

1 **MB** = Member body (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
NOTE Columns 1, 2, 4, 5 are compulsory.

Template for comments and secretariat observations

Date: 2016-10-15	Document: **SC 22 N 5131**	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 1		1.1	2	ed	It is proposed that "C++17 should refer to C11 instead of C99" in P0063 and this proposal is accepted. So it needs to change the base C programming language to C11 from C99.	C++ is a general purpose programming language based on the C programming language as described in ISO/IEC 9899:~~1999~~ 2011 *Programming languages — C*	
JP 2		3.2	6	ed	The subclause , "The inline specifier", was added by P0836 and the description of inline function was moved to this subclause. So it needs to change the reference to 7.1.6[dcl.inline] from 7.1.2[dcl.fct.spec]. In addition, it needs to add the reference of `inline variable with external linkage'.	There can be more than one definition of a class type (Clause 9), enumeration type (7.2), inline function with external linkage (~~7.1.2~~ 7.1.6) , inline variable with external linkage(7.1.6),	
JP 3		3.7	2	ed	`operator new' should be replaced by `new-expression'	The dynamic storage duration is associated with objects created with ~~operator new~~ new-expression	
JP 4		3.8	(6.5)	ed	&pb mismatches the comment.	~~&~~*pb; *// OK:* pb *points to valid memory*	
JP 5	6	4.4	1/Example	ed	A semicolon is required at the end.	struct X { int n; };	
JP 6		5.17	2	ed	"function returning T" which was modified to "function type T" was enclosed in double quotes, but "function type T" was not enclosed in double quotes. (In this sentence, "function type T" is in apposition to "array of T" and "array of T" is enclosed in double quotes, but "function type T" is not.) So it needs to enclose "function type T" in double quotes.	from "array of T" or "function type T" to "pointer to T".	
JP 7		8.3.5	5	ed	The same as the comment for 5.17/2.	any parameter of type "array of T" or of "function type T" is adjusted to be "pointer to T".	

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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 8		8.4.1	2	ed	The paragraph was modified to fix C++ standard core issue 2145(http://www.open-std.org/Jtc1/sc22/wg21/docs/cwg_active.html#2145). Fixing the issue itself is good, but the new phrase doesn't look correct. "`void declarator ;`" and "`declarator ;`" are enumerated, but the former constitutes a function definition and the latter does not.	Drop the paragraph. Or, simply "The form of *declarator* is described in 8.3.5."	
JP 9		8.4.3	4	ed	The same as the comment for 3.2/6.	A deleted function is implicitly an inline function (~~7.1.2~~7.1.6).	

JP 10		9.2	7	ed	A space is not needed after `T'.	struct S {
						using T = void();
						T * p = 0; *// OK: brace-or-equal-initializer*
						virtual T f = 0; *// OK: pure-specifier*
						};	

JP 11		9.4	1	ed	`0' should be replaced by `nullptr`.	local* p = ~~0~~ nullptr; *// error:* local *not in scope*	
JP 12		10.1	7/Figure 4 — Virtual base	ed	"Figure 4 — Virtual base" is referred to from 10.1/6 but located in 10.1/7. It's confusing for readers.	Move figure 4 to inside 10.1/6.	
JP 13		11.3	7	ed	The same as the comment for 3.2/6.	Such a function is implicitly an inline function (~~7.1.2~~7.1.6).	
JP 14		14.1	8	ed	The same as the comment for 5.17/2.	A non-type *template-parameter* of type "array of T" or of "function type T" is adjusted to be of type "pointer to T".	
JP 15		15.2	5	ed	This deallocation function includes the class deallocation function. (There is the reference to 12.5[class.free] in the language specification of C++14.) So it needs to add the reference to 12.5[class.free].	If the object was allocated by a *new-expression* (5.3.4), the matching deallocation function (3.7.4.2, 12.5), if any, is called to free the storage occupied by the object.	

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

Template for comments and secretariat observations

Date: 2016-10-15	Document: **SC 22 N 5131**	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 16		15.3	2	ed	The same as the comment for 5.17/2.	A handler of type "array of T" or "function type T" is adjusted to be of type "pointer to T".	
JP 17		15.4	2	ed	The same as the comment for 5.17/2.	A type *cv* T denoted in a *dynamic-exception-specification* is adjusted to type T. A type "array of T", or "function type T" denoted in a *dynamic-exception-specification* is adjusted to type "pointer to T".	
JP 18		16.1	8	ed	The footnote #148 is across two pages.	Locate all #148 sentences in a single page.	
JP 19		16.8	1	te	It describes "__cplusplus function is defined to the value 201402L". The value means C++14, so it should be changed in C++17	Change 201402L to something appropriate like 2017xx.	
JP 20		18.6.4		te	The name std::launder() seems cryptic at least for non-English native speakers. There is no hint in the word "launder" to show it is about the C++ object model, lifetime, and reusing storage. The situation is likely same even if a programmer preliminarily knows about the issues it solves. Comments like "Here, compilers should suppose new object at reused storage" will be wanted each time it is used. The following function names are better. - reuse_existing_storage - suppose_new_at_reused_storage … The changes of the label of this chapter and sample codes are accompanied by this change.	template <class T> constexpr T* ~~launder~~ reuse_existing_storage(T* p) noexcept;	

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

Template for comments and secretariat observations

Date: 2016-10-15	Document: **SC 22 N 5131**	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
JP 21		25		ed	The order of *Requires*, *Effects* and *Returns* sections for each function templates are not consistent in this clause. For some templates, *Requires* comes after *Effects* and even after *Returns*. It would be better to describe in a consistent manner.	Reorder the sections for each algorithm templates in the same order, as *Requires*, *Effects* and *Returns*.	
JP 22		25.3.10	2	ed	*j* is defined but not used in (2.2) and (2.3). Some parts of expressions can be replaced with the *j*.	(2.2) "`!(*i == *j)`" (2.3) "`pred(*i, *j) == false`"	
JP 23		25.4.1		ed	`std::copy_backward` and some other algorithms don't have parallelized versions. We can know from the list in 25.1 which algorithms have them, but it would be better to specify in each description explicitly.	Add "*Remarks*: No parallel algorithm overload is available." for each algorithm that doesn't have its parallelized overload.	
JP 24		25.5.10	1	ed	*Effects* section for `std::next_permutation` describes about the return value, too. But it should be in *Returns* section as in `std::prev_permutation`.	Replace the 3rd and 4th sentences with a new paragraph "*Returns:* `true` if such a permutation exists. Otherwise, it transforms the sequence into the smallest permutation, that is, the ascendingly sorted one, and returns `false`."	
JP 25		26.5.7	9	ed	Parameter theta of polar has the type of the template parameter. Therefore, it needs to change the default initial value to T(). The change of the declaration of this function in 26.5.1 is accompanied by this change.	template<class T> complex<T> polar(const T& rho, const T& theta = ~~0~~T());	
JP 26		26.8.5	1	ed	There is a typo in the parameter of the second declaration. (gterator instead of Iterator)	template <class InputIterator1, class InputIterator2, class T, class BinaryOperation1, class BinaryOperation2> T inner_product(InputIterator1 first1, InputIterator1 last1, Input~~g~~Iterator2 first2, T init, BinaryOperation1 binary_op1, BinaryOperation2 binary_op2);	

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

Template for comments and secretariat observations

Date: 2016-10-15	Document: **SC 22 N 5131**	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 27		27.11.1		te	In C11- ISO/IEC 9899:2011(E), formatted input/output functions (with '_s' suffix) are added as annex K.3.5.3. Those functions promote safer, more secure programming because they verify that output buffers are large enough for the intended result and return a failure indicator if they are not. Data is never written past the end of an array. All string results are null terminated. Those functions also benefit C++. We propose to add them to C++17.	Add the functions defined in the subclauses of C11 K.3.5.3.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CA 1	all	18.10.5 18.3.2.4 18.5 18.9 20.2.1 20.2.4 20.14	all	te	P0270R1 went through SG1 and LWG but was too late to make it to the straw polls. The problems it addresses stem from referring to C11, which came into C++17 at the last minute. P0270R1 should have made it in with the C11 change.	Apply all of P0270R1, "Removing C dependencies from signal handler wording", to C++17.	
CA 2	all	27.10.8.1 [path.generic]	all	te	*root-name* is effectively implementation-defined. As acknowledged by the note under *root-name* in the grammar, //is an example of what a *root-name* may be. Should *root-name* be // for a specific implementation, the grammar is ambiguous. The string //a may resolve as either *root-name root-directory*$_{opt}$ *relative-path*$_{opt}$ *//root-directory*$_{opt}$ *relative-path*$_{opt}$ *//relative-path*$_{opt}$ *//filename* *//name*	Change under *root-name* in the grammar of subclause 27.10.8.1 [path.generic]: An implementation-defined path prefix**operating system dependent** name that identifies the starting location for absolute paths. Add a new paragraph before paragraph 1 of [path.generic]: The *root-name* in a *pathname* is the longest sequence of characters that could possibly form a *root-name*.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					//a or *root-directory relative-path*$_{opt}$ *directory-separator relative-path*$_{opt}$ *slash directory-separator relative-path*$_{opt}$ */directory-separator relative-path*$_{opt}$ */slash relative-path*$_{opt}$ *//relative-path*$_{opt}$ *//filename* *//name* //a		
CA 3	all	27.10.8 [class.path]	all	te	The term “pathname” in 27.10.8 [class.path] is ambiguous in some contexts. For details refer to P0430R0 section 2.1.	Add the following specification to 27.10.8.2.1 [path.fmt.cvt]: Specifications for path appends, path concatenation, path modifiers, path decomposition and path query are in terms of the generic pathname format. An implementation needs to make whatever changes necessary to the	

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

Template for comments and secretariat observations	Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						pathname in native pathname format to produce the specified change in the generic pathname format, or return query result for pathname in terms of the generic pathname format.	
CA 4	all	27.10.8.4.1 [path.construct]	all	te	Extra flag in path constructors is needed to distinguish whether source is in native pathname format, or generic pathname format. For details refer to P0430R0 section 2.2.	Refer to P0430R0 section 2.2.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CA 5	all	27.10.8.1 [path.generic]	all	te	*root-name* definition is over-specified. The description of *root-name* limits its use to be the starting location for absolute paths. This is overly restrictive and disregards established practice where special prefixes on path names is treated as a trigger for alternate path resolution on certain operating systems. There are cases where such alternative path resolution relies on context from the environment such as the identity of the current user; therefore, the presence of a special prefix on a path name is not always indicative of an absolute path. For details refer to P0430R0 section 2.3.1.	Modify root-name definition in 27.10.8.1 [path.generic]: root-name: An operating system dependent name that ~~identifies the starting location for absolute paths~~ can be used to disambiguate the remainder of the path. [Note: A root-name can be used to identify the starting location for absolute paths; it can also be used to invoke alternative pathname resolution. Many operating systems define a name beginning with two directory-separator characters as a root-name that identifies network or other resource locations. Some operating systems define a single letter followed by a colon as a drive specifier – a root-name identifying a specific device such as a	

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: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						disk drive. —end note]	
CA 6	all	27.10.8.4.3 [path.append]	all	te	Operator/ (and other append) semantics not useful if argument has *root-name*. A non-POSIX operating system could design its generic pathname for native file type to have a *root-name* and use it in some creative way. For example, if argument p has a *root-name*, then p's *root-name* have to be removed before appending.	Refer to P0430R0 section 2.3.2.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					For details refer to P0430R0 section 2.3.2.		
CA 7	all	27.10.15.1 [fs.op.absolute]	all	te	Member function absolute in 27.10.4.1 is over-specified for non-POSIX-like operating system. For details refer to P0430R0 section 2.4.1.	Modify the specification of absolute function in 27.10.15.1 [fs.op.absolute]: … Returns: ~~An absolute path (27.10.4.1) composed according to Table 122.~~ If status(p).type() is an implementation-defined file type, then the returned path is implementation-defined. Otherwise, an absolute path (27.10.4.1) composed according to Table 122. ...	
CA 8	all	27.10.13 [class.directory_iterator] 27.10.15.3	all	te	Some file system operation functions are over-specified for implementation-defined file type. For details refer to P0430R0 section 2.4.2.	Refer to P0430R0 section 2.4.2.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
		[fs.op.copy] 27.10.15.14 [fs.op.file_size] 27.10.15.35 [fs.op.status]					
CA 9	all	all	all	ge	The present references to UCS2 in the Committee Draft are appropriate in the interests of preventing silent breakage of software written to older versions of C++.	Preserve the references to UCS2 as presented in the Committee Draft.	
CA 10	all	all	all	ge	The adoption of the changes proposed in WG21 document P0292R2 (constexpr if-statements) is a step in the right direction.	Preserve the functionality as presented in the Committee Draft.	
CA 11	all	1.8 [intro.object]	paragraph 3	te	Relative to C++14, this CD introduces additional special behaviour for `unsigned char`. This is	● Adopt P0257R1, “A byte type for increased type safety”, with necessary	

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: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					harmful to optimizing existing code, and we would like to avoid this unwanted outcome.	changes from WG21 review. ● To minimize scope, rename std::byte to std::storage_byte (or std::raw_byte). This also avoids confusion, as the proposed std::byte does not match existing common uses of the word 'byte'. Using 'byte' as suggested in P0257R1 would go against "standardizing existing practice". ● Modify 1.8 [intro.object] paragraph 3 by replacing "array of *N* `unsigned char`" with "array of *N* `std::storage_byte`" (or `std::raw_byte`). Adjust examples and notes accordingly.	
CA 12	all	1.8 [intro.object] 3.10 [basic.lval]	various	te	The status of the following code should be explicitly indicated in the Standard to avoid surprise: `#include <new>`	Include an example (and complimentary notes) indicating that the code presented has undefined behaviour for the reasons set out herein.	

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/ NC1	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment2	Comments	Proposed change	Observations of the secretariat

					`int bar() {`
					`  alignas(int) unsigned char space[sizeof(int)];`
					`  int *pi = new (static_cast<void *>(space)) int;`
					`  *pi = 42;`
					`  return [=]() mutable { return *std::launder(reinterpret_cast<int *>(space)); }();`
					`}`
					
					In particular, it appears that the call to `std::launder` has undefined behaviour because the captured copy of `space` is not established to provide storage for an object of type `int` (subclause 1.8 [intro.object] paragraph 1).
					Furthermore, the code has undefined behaviour also because it attempts to access the stored value of the `int` object through a glvalue of an		

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					array type other than one of the ones allowed by subclause 3.10 [basic.lval] paragraph 8.		
CA 13	all	all	all	ge	As the Committee Draft has already been shipped, the addition of further major features (e.g., operator dot, subset of the Concepts TS, std::exception_list, default comparison operators) will likely destabilize the document and reduce consensus.	WG21 is requested to commit to the status quo of the CD except where there is overwhelming consensus in support of specific changes. Where there is a lack of overwhelming support for general categories of changes, WG21 is requested to commit to the status quo of the CD.	
CA 14	all	20.11.2.2	4	te	The removal of the "debug only" restriction for use_count() and unique() in shared_ptr introduced a bug: in order for unique() to produce a useful and reliable value, it needs a synchronize clause to ensure that prior accesses through another reference are visible to the successful caller of unique(). Many current implementations use a relaxed load, and do not provide this guarantee, since it's not stated in the Standard. For debug/hint usage that was OK. Without it the specification is unclear and	A solution could make `unique()` use `memory_order_acquire`, and specifying that reference count decrement operations synchronize with `unique()`. This won't provide sequential consistency but may be useful. We could also specify `use_count()` as only providing an unreliable hint of the actual count, or deprecate it.	

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

Template for comments and secretariat observations

Date: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					misleading.		
CA 15	all	16.8	1	te	`__cplusplus` is defined to the value `201402L`.	Update to a date reflecting the expected ratification year / month.	
CA 16	all	20.11.2.6 29.6.5	all	te	The resolution to LWG2445 "'Stronger' memory ordering" was lost between SG1 and LWG. The technical issue is minor but often confuses developers, it would be unfortunate to avoid resolving it for C++17.	Implement a solution along the lines of p0418r1.	
CA 17	all	25.2.4	all	ge	The behavior of parallel algorithms when an exception leaves the algorithm is to call std::terminate. This behavior does not prevent developers from throwing exceptions, as long as these exceptions are caught. The behavior has desirable performance effects for parallel algorithms. This behavior matches that of std::thread and main when exceptions leave them. It can be	Preserve the functionality from p0394r4, as adopted in the Committee Draft.	

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: 2016-10-02	Document: CA comments for SC22 N5131	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					augmented with policies or executors in future versions of the Standard without breaking backwards compatibility with C++17. Notably, some form of exception list can be added to the Standard. In the meantime, developers can implement their own exception list in C++17, which would help the committee standardize their existing practice.		
CA 18	all	all	all	ge	The Committee Draft has already been shipped, and the proposal in p0145 was heavily reviewed in Oulu. Departure from consensus reached for p0145 on expression evaluation order will likely destabilize the document and reduce consensus. In particular, discussions about performance impact on user code as well as general correctness of user code in the face of expression evaluation order affected voting on p0145.	WG21 is requested to commit to the consensus reached for p0145 in Oulu plenary, except when changes to expression evaluation order for C++17 would be in the details and supported with solid technical reasoning, including performance evaluation on multiple implementations. Changes in the scope of the proposal should be postponed until after C++17.	

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

ISO/IEC CD 14882 Comments Template

Date: 2016-09-05	Document: **SC22 N5130**	Project: 14882

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
FI 1				te	All open Core Issues should be resolved.	As CWG sees fit.	
FI 2				te	All open Library Issues should be resolved.	As LWG sees fit.	
FI 3		8.5		te	Decomposition declarations do not allow specifying the type of the identifiers introduced. This is inconsistent with every other mechanism for introducing an identifier, and makes large-scale programming harder.	Either provide a language syntax for specifying the type of the identifiers, or provide a library facility for enforcing the type.	
FI 4		14.9		te	Deduction guides are not integrated to the standard library. Early attempts to do so have revealed that implicit deduction guides easily lead to deducing class template arguments as references in surprising places, and that implicit deduction guides make as-if refactorings of library interfaces harder; such refactorings that used to be non-detectable now become breaking changes when implicit deduction guides can be used. Deduction guides can't be deleted when the user wants to turn off certain kinds of deduction; the proposed work-around is changing the class template definition, which is rather hard for code that the user doesn't own. Explicit deduction guides are ambiguous with implicit ones if both match, which makes post-hoc adaptation hard or impossible.	We should explore ways to make the semantics of deduction guides less error-prone, and add explicit deduction guides to the library where applicable.	
FI 5				te	The proposal p0067, Elementary string conversions was accepted for C++17 but not incorporated due to seemingly minor problems in the specification. Those problems have since been fixed by a follow-up paper, and the facility should be incorporated into C++17.	Consider the latest version of the proposal to be incorporated into C++17.	
FI 6		21.4			The class template string_view was adopted into the working draft without the corresponding user-defined literal. Such literals have been implemented as extensions.	Add a user-defined literal for string_view.	
FI 7		20		te	The proposal p0032 has multiple problems: 1) it turns member function .empty() into .has_value(),	Keep the .empty() functions (and introduce them to all the types that are supposed to have a	

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: 2016-09-05	Document: **SC22 N5130**	Project: 14882

MB/ NC[1]	Line number (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					negating the logic. Refactoring e.g. existing uses of std::experimental::any to use std::any thus involve non-trivial refactorings that are error-prone and can't be done via simple search-and-replace if there are containers in the same source files for which .empty() is used (based on the implementation experience of making the change in libstdc++ and refactoring the testsuite). Whilst any is not a container, the library is failing to go towards a direction where there would be a generic way to query for emptiness. 2) The use of function references for tag types makes the interface hard to use. The tag types do not have value semantics like every other tag type has, the tag types are hard to construct, and present surprises for certain kinds of overload sets. Furthermore, any attempts to decay the tag types produces a really surprising effect – as opposed to what the other tag types do, which is that the result of decaying them is the tag type itself, decaying these new tag types results in a pointer to function.	homogeneous interface), and make the tag types be regular tag types that are not references to functions.	
FI 8		30.4.2.1		te	The class template lock_guard was made variadic. This is abi-breaking, and confusing because one-argument lock_guards have a typedef mutex_type but lock_guards with more than one argument don't. There's no need to try to shoehorn this functionality into one type.	Revert the changes to lock_guard, and introduce a new variadic class template vlock_guard that doesn't have the mutex_type typedef at all.	
FI 9		20, 30		te	The variables of library tag types need to be inline variables. Otherwise, using them in inline functions in multiple translation units is an ODR violation.	Make piecewise_construct, allocator_arg, nullopt, (the in_place_tags after they are made regular tags), defer_lock, try_to_lock and adopt_lock inline.	
FI 10		20.6		te	Adopt the proposed resolution of LWG 2756 into C++17, to provide converting constructors and assignment operators for optional.	Adopt the latest proposed resolution of LWG 2756, which should be available by Issaquah.	
FI 11		20.8		te	Adopt the proposed resolution of LWG 2744 and 2754 so that std::any can't be made to hold non-	Adopt the proposed resolution of LWG 2744 and 2754.	

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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					copyable types.		
FI 12		20.8		te	Adopt the proposed resolution of LWG 2509, which allows any_cast to move when it can.	Adopt the proposed resolution of LWG 2509 into C++17.	
FI 13		20		te	Adopt the proposed resolution of LWG 2729, which makes pair and tuple constructors and assignment operators reflect the well-formedness of the constructors and assignment operators of the elements.	Adopt the proposed resolution of LWG 2729.	
FI 14		27.10.12.3		te	LWG 2761 should be resolved and the resolution adopted into C++17, in order to make directory_entry comparisons non-members, so as to allow conversions on both sides of the comparison, which is consistent with other such operators in the library.	Make the comparison operators of directory_entry non-members.	
FI 15		20.6		te	The hash specialization of optional should be a “poison type” if there is no valid hash for the element type of optional.	Adopt a solution similar to LWG 2543 for optional’s hash.	
FI 16		20, 23		te	Relational operators for containers should sfinae; if the underlying type is not comparable, neither should the container be. Same applies to tuple and pair.	Make the relational operators of containers and utility components reflect the validity of the underlying element types.	
FI 17		20, 23		te	The relational operators of optional and variant completely reflect the semantics of the element types; this is inconsistent with other types in the library, like pair, tuple and containers. If we believe it’s important that we don’t synthesize relational operators for wrapper types, we should believe it’s important for other types as well. Otherwise comparing containers of floating-point types and tuples/pairs etc. of floating point types will give incorrect answers.	Make the relational operators of containers and utility components reflect the semantics of the operators for the underlying element types.	
FI 18		20.14.15			It was thought that using default_order as the default comparison for maps and sets was not abi-breaking but this is apparently not the case.	Revert the change to the default comparison of maps and sets.	

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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
FI 19		20.10		te	The changes in the paper p0414 should be adopted into C++17.	Adopt the changes in p0414.	
FI 20		8.5		te	Decomposition declarations do not allow parentheses-syntax; auto [a, b, c](expr); is not valid, which is syntactically inconsistent with non-decomposition declarations.	Allow using parentheses in decomposition declarations.	
FI 21		14.9		te	Class templates can't be constructed with brace-syntax when class template argument deduction for constructors is used; templatename{a,b,c} is not valid.	Allow using braces in such initialization.	
FI 22		20.7		te	Is it intentional that variant can "hold" a void? Chances are that it's useful for using variant as a typelist, so we're not recommending changing that at this point, so this comment is purely to allow discussion about this aspect.		
FI 23		8.5		te	Nested decomposition declarations can't work, as they clash with the attribute syntax.	Consider changing the syntax for decomposition declarations, or fixing the problem some other way.	

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** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
CH 1		all		ge	The active issues on the issues lists shall be addressed before the standard becomes final. The higher frequency of standard revisions should not be an excuse for more bugs.		
CH 2		1.9 [intr.execution]		te	Clarify `volatile`	Adopt a resolution discussed on the reflector.	
CH 3		20.6 [optional], 20.7 [variant], 20.8 [any]		te	The new `in_place` tags prevent perfect forwarding. They decay to function pointers, at which point they are no longer tags. This makes programming with them a burden, while the intent was to simplify it by re-using a common name.	Re-introduce `in_place_t`/`in_place`, `in_place_type_t<T>`/`in_place_type<T>`, `in_place_index_t<I>`/`in_place_index<I>` by reverting this specific part of p0032r2.	
CH 3		20.7 [variant]		te	`variant` allows reference types as alternatives; `optional` explicitly forbids to be instantiated for reference types. This is inconsistent.	Consider allowing reference types for both or none.	
CH 4		20.7.2 [variant.variant]		te	`variant<int,void>` should be as usable as `variant<int>`		

CH 5		20.7.2 [variant.variant]		te	`variant<>` should not have an `index()` function	Consider specifying a specialization for `variant<>` like:
						
						`template<> class variant<> {`
						`public:`
						`  variant() = delete;`
						`  variant(const variant&)`
						`    = delete;`
						`  variant&`
						`  operator=(variant const&)`	

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 (e.g. 17)	Clause/ Subclause (e.g. 3.1)	Paragraph/ Figure/ Table/ (e.g. Table 1)	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

						`    = delete;`
						`};`	

CH 6		20.7.2 [variant.variant]		te	Clarify the intended behavior of `variant` for alternative types that are references.	Add a respective note.	
CH 7		20.7.2 [variant.variant]		te	Consider making the variant statically `!valueless_by_exception()` for cases where `is_nothrow_move_constructible_v<T_i>` for all alternative types `T_i`	Adopt section III of P0308R0.	
CH 8		20.7.2.1 [variant.ctor]		te	Clarify `variant` construction.	Add a note that `variant<>` cannot be constructed.	
CH 9		21.4 [string.view]		te	The standard library should provide `string_view` parameters instead or in addition for functions defined with `char const *` or `string const &` as parameter types. Most notably in cases where both such overloads exist or where an internal copy is expected anyway. It might be doubted that the non-null termination of `string_view` could be an issue with functions that pass the `char *` down to OS functions, such as `fstream_buf::open()` etc and those shouldn't provide it and favour generating a `std::string` temporary instead in that case. However, `std::path` demonstrates it is	Provide the overloads for `std::regex`, the exception classes, `std::bitset`, `std::locale` and more.	

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:	Document:	Project:

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
					usable to have `string_view` overloads and there might be many places where it can be handy, or even better.		
CH 10		25.2.3 [algorithms.parallel.exec]		te	Parallel implementations of algorithms may be faster if not restricted to the complexity specifications of serial implementations.	Add a relaxation of complexity specifications for non-sequenced policies.	
CH 11		25.2.3 [algorithms.parallel.exec]		te	It may be useful to copy objects to a separate space for non-sequenced policies.	Add explicit allowance for non-sequenced policies to copy the objects they work on.	

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