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

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

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