



ISO/IEC JTC 1/SC 22 "Programming languages, their environments and system software interfaces"

Secretariat: ANSI

Committee manager: Ash Bill Mr



SoV and Collated Comments - ISO_IEC CD 14882

Document type	Related content	Document date	Expected action
Ballot / Result of voting		2025-10-02	INFO

Description

The CD consultation of 14882 closed with comments from Austria, Brazil, Bulgaria, Canada, China, Czech Republic, Finland, France, Germany, Italy, Netherlands, Poland, Romania, Russian Federation, Spain, Sweden, Switzerland, United Kingdom, and United States. The CD results and the accompanying comments have been forwarded to the Project Editor and SC 22/WG 21 for review and resolution of the comments. The Project Editor is instructed to prepare an approved Disposition of Comments document and a revised text for further processing.

Result of voting

Ballot Information

Ballot reference	ISO/IEC CD 14882
Ballot type	CD
Ballot title	Programming languages — C++
Opening date	2025-08-06
Closing date	2025-10-01
Note	

Member responses:

Votes cast (26)	Austria (ASI) Brazil (ABNT) Bulgaria (BDS) Canada (SCC) China (SAC) Czech Republic (UNMZ) Denmark (DS) Finland (SFS) France (AFNOR) Germany (DIN) India (BIS) Ireland (NSAI) Israel (SII) Italy (UNI) Japan (JISC) Kazakhstan (CTRM) Korea, Republic of (KATS) Netherlands (NEN) Poland (PKN) Romania (ASRO) Russian Federation (GOST R) Spain (UNE) Sweden (SIS) Switzerland (SNV) United Kingdom (BSI) United States (ANSI)
Comments submitted (1)	Türkiye (TSE)
Votes not cast (1)	Ukraine (SE UkrNDNC)

Questions:

Q.1	"Do you have any comments related to the Committee Draft?"
------------	--

Votes by members	Q.1
Austria (ASI)	Yes

Brazil (ABNT)	Yes
Bulgaria (BDS)	Yes
Canada (SCC)	Yes
China (SAC)	Yes
Czech Republic (UNMZ)	Yes
Denmark (DS)	Abstain
Finland (SFS)	Yes
France (AFNOR)	Yes
Germany (DIN)	Yes
India (BIS)	Abstain
Ireland (NSAI)	No
Israel (SII)	Abstain
Italy (UNI)	Yes
Japan (JISC)	No
Kazakhstan (CTRM)	Abstain
Korea, Republic of (KATS)	No
Netherlands (NEN)	Yes
Poland (PKN)	Yes
Romania (ASRO)	Yes
Russian Federation (GOST R)	Yes
Spain (UNE)	Yes
Sweden (SIS)	Yes
Switzerland (SNV)	Yes
United Kingdom (BSI)	Yes
United States (ANSI)	Yes

Answers to Q.1: "Do you have any comments related to the Committee Draft?"

19 x	Yes	Austria (ASI) Brazil (ABNT) Bulgaria (BDS) Canada (SCC) China (SAC) Czech Republic (UNMZ) Finland (SFS) France (AFNOR) Germany (DIN) Italy (UNI) Netherlands (NEN)
------	-----	---

		Poland (PKN) Romania (ASRO) Russian Federation (GOST R) Spain (UNE) Sweden (SIS) Switzerland (SNV) United Kingdom (BSI) United States (ANSI)
3 x	No	Ireland (NSAI) Japan (JISC) Korea, Republic of (KATS)
4 x	Abstain	Denmark (DS) India (BIS) Israel (SII) Kazakhstan (CTRM)

Comments from Voters		
Member:	Comment:	Date:
Austria (ASI)	<i>Comment File</i>	2025-09-22 06:33:43
CommentFiles/ISO_IEC CD 14882_ASI.docx		
Brazil (ABNT)	<i>Comment</i>	2025-10-01 19:16:32
Adopt the bug fix wording proposed in P3579R2 "Fix matching of constant template parameters when matching template template parameters" (or latest revision) into the working draft.		
Bulgaria (BDS)	<i>Comment File</i>	2025-09-18 18:57:12
CommentFiles/ISO_IEC CD 14882_BDS.doc		
Canada (SCC)	<i>Comment File</i>	2025-09-17 21:52:02
CommentFiles/ISO_IEC CD 14882_SCC.doc		
China (SAC)	<i>Comment</i>	2025-09-15 08:28:35
<p># Trivial relocatability should imply bitwise relocatability</p> <p>**Submitter:** Mingxin Wang</p> <p>**Related paper:** P2786R13: Trivial Relocatability For C++26</p> <p>## Summary of Concern</p> <p>The current definition of "trivially relocatable" in P2786R13 does not guarantee bitwise relocatability (i.e., that an object can be relocated via <code>memcpy</code>). This significantly weakens the utility of the feature, especially for performance-critical applications such as type erasure, and creates a gap that other proposals (e.g., P3780R0 <code>is_bitwise_trivially_relocatable</code>) are trying to fill. We believe that "trivially relocatable" is the correct and intuitive place for this guarantee.</p>		

Proposed Wording Changes

Modify 11.2 [class.prop] to reflect the bitwise copy requirement and implementation-defined behavior for special pointers.

Change the definition of when a class is *eligible for trivial relocation* as follows:

```diff

A class is \*eligible for trivial relocation\* unless it

- has any virtual base classes,
- has a base class that is not a trivially relocatable class,
- has a non-static data member of an object type that is not of a trivially relocatable type, or
- + — has a subobject of a type where a bitwise copy may not preserve the semantics of the object (e.g., types with pointer authentication), or
- has a deleted destructor,

except that it is implementation-defined whether an otherwise-eligible union having one or more subobjects of polymorphic class type is eligible for trivial relocation.

```

Detailed Comments

1. The "Trivial" in "Trivially Relocatable" Should Imply Bitwise Operation

The term "trivial" in C++ has a strong precedent for implying bitwise operations (e.g., trivially copyable). When a developer sees "trivially relocatable", the natural and overwhelming expectation is that the object can be moved with `memcpy`. Deviating from this expectation makes the feature less intuitive and harder to use correctly.

The argument that certain architectures or compiler optimizations (like pointer authentication) might require more than a `memcpy` is noted. However, we believe this concern is misplaced for a "trivial" trait. If a type requires special handling for relocation, it should not be considered "trivially relocatable", in the same way that a polymorphic type with a `vptr` is not trivially copyable. The complexity should disqualify the type from being "trivial," rather than watering down the meaning of "trivial" for all users.

2. Critical Need for Bitwise Relocatability in Type Erasure

Type erasure libraries are a primary motivator for this feature. These libraries manage objects of unknown types through a common interface. For performance, they need to be able to relocate their owned objects in memory (e.g., when a `std::vector`-like buffer reallocates) using `memcpy`. Without a guarantee of bitwise relocatability, these libraries face a dilemma:

* **Assume `memcpy` is safe:** This is what many libraries do today for types that are `std::is_trivially_copyable_v` && `std::is_trivially_destructible_v`, but it is not guaranteed to be correct for all relocatable types under the proposed definition. This leads to subtle, hard-to-debug issues on platforms where the assumption fails.

* **Use the move constructor:** This is safer but incurs a significant performance penalty, defeating one of the main purposes of introducing a relocation mechanism.

Since type erasure facilities operate without compile-time type information, they cannot check for specific types that might require special relocation logic. They need a single, reliable query (`is_trivially_relocatable`) that guarantees `memcpy` is safe.

3. Proposed Solution

We propose that the definition of "trivially relocatable" be strengthened to guarantee bitwise relocatability. This ensures that `std::is_trivially_relocatable_v<T>` provides the strong, portable guarantee that developers need: if it is `true`, `memcpy` is safe.

To address the concerns about special pointer types (e.g., with pointer authentication), we suggest the following:

**Make it implementation-defined whether types with special pointer semantics (like those requiring pointer authentication) are considered `is_trivially_relocatable`.

This approach has precedent. The behavior of many aspects of the C++ memory model and type system is implementation-defined, allowing vendors to make choices appropriate for their platforms. An implementation where pointers require special handling upon relocation would simply not mark types containing them as trivially relocatable. This puts the burden of complexity on the implementation, where it belongs, rather than on the user.

This makes the feature significantly more valuable and less error-prone for the entire C++ community.

Czech Republic (UNMZ)	Comment File	2025-09-30 16:40:21
CommentFiles/ISO_IEC CD 14882_UNMZ.doc		
Finland (SFS)	Comment	2025-09-23 08:51:25
Comments attached		
Finland (SFS)	Comment File	2025-09-23 08:51:25
CommentFiles/ISO_IEC CD 14882_SFS.docx		
France (AFNOR)	Comment	2025-09-23 18:06:56
Please find attached the French comments.		
France (AFNOR)	Comment File	2025-09-23 18:06:56
CommentFiles/ISO_IEC CD 14882_AFNOR.docx		
Germany (DIN)	Comment File	2025-10-01 13:13:31
CommentFiles/ISO_IEC CD 14882_DIN.docx		
Italy (UNI)	Comment File	2025-09-15 12:07:08
CommentFiles/ISO_IEC CD 14882_UNI.doc		
Netherlands (NEN)	Comment	2025-09-19 14:12:19
<p>- Contracts as present in C++26 are vital for future development of a plurality of safety features, including functional safety and memory safety. Even if the US government has dropped its attention from our language, we should not use that as an excuse to drop the ball and to forget that there are still many preventable problems.</p> <p>- The function evaluation_exception specified in the contract_violation objects created by the contract violation handler may have a security risk on one platform, and offers insufficient tangible benefit for the risk it causes. We propose to remove it from C++26 to potentially be added back in '29 if we can show that the security risk does not actually materialize on that platform. See also P3819.</p>		
Poland (PKN)	Comment File	2025-09-24 11:09:55
CommentFiles/ISO_IEC CD 14882_PKN.docx		
Romania (ASRO)	Comment File	2025-10-01 12:52:03
CommentFiles/ISO_IEC CD 14882_ASRO.docx		

Russian Federation (GOST R)	Comment File	2025-08-27 17:30:17
CommentFiles/ISO_IEC CD 14882_GOST R.doc		
Spain (UNE)	Comment File	2025-09-30 11:52:18
CommentFiles/ISO_IEC CD 14882_UNE.doc		
Sweden (SIS)	Comment	2025-10-01 09:39:00
There are several open points regarding the newly added chapter on contract assertions. The considerations of our NB are summarized in P3849R0, which has been shared with WG21: https://isocpp.org/files/papers/P3849R0.pdf We hope that the topics raised will be addressed accordingly.		
Switzerland (SNV)	Comment	2025-09-24 08:15:08
The Committee Draft proposes in clause 33 [exec] components for asynchronous execution of function objects. The asynchronous execution might be triggered by asynchronous signals. However, the existing components in clause 33 [exec] don't provide any mechanism to trigger such execution in a signal-safe way. This is a serious defect. This defect needs to be fixed before the standard is published, e.g. by adopting a mechanism as proposed in P3669.		
United Kingdom (BSI)	Comment File	2025-09-25 10:07:17
CommentFiles/ISO_IEC CD 14882_BSI.doc		
United States (ANSI)	Comment File	2025-09-23 21:02:42
CommentFiles/ISO_IEC CD 14882_ANSI.docx		

Comments from Commenters		
Member:	Comment:	Date:
Türkiye (TSE)	Comment	2025-09-30 09:47:13
-		

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
NC IT- 002	-	-	-	Te	P1789 adds support for destructuring <code>std::integer_sequence</code> into a pack, completing the design of P1061: Structured Bindings can introduce a Pack. To the best of our knowledge, this paper already has a strong consensus, and has an extremely narrow scope and minimum impact on the standard document.	Adopt P1789	
US 2- 404		general		ed	There are two occurrences of “ <i>compound-statement</i> of an expansion statement” and two of <i>compound-statement</i> of an <i>expansion-statement</i> ”.	Decide on the correct wording and use the same throughout.	
US 1- 405		general		ed	The tuple representing direct base class relationships is sometimes in code font (e.g., 6.7 bullet 16.4.2; 11.7.1 paragraph 2) and sometimes not (e.g., 21.4.6 bullet 2.5).	Decide on one style and apply it throughout.	
GB01 -013		1	2	ed	Consider updating comparison to C While not wrong, the list of features C++ provides over and above C is essentially unchanged since C++98. Given how much C++ has changed since then, this list undersells the language by omitting many key distinguishing features. It is also the only place other than the index to use the term "free store"	Consider updating this to include features that differentiate the latest Draft C++ Standard from the normative reference version of the C Standard (ISO/IEC 9899:2024). (Acknowledging that 1. C is a moving target, and 2. what goes here is inherently subjective and only ever the tip of the iceberg). Consider adding (e.g.): - type safety and concepts - automatic resource management - contracts - modules - compile-time reflection and metaprogramming	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						Consider replacing "free store management operators" with "type-aware allocation and deallocation operators," or "automatic memory management facilities," or removing in favour of the existing "library facilities" already covering recommended modern memory management mechanisms.	
FR-001 -014		4.1	8	te	The hardened implementation should be specified independently of contracts which are not a proven technology. All existing deployed field experience are not based on contracts.	Change "contract violation" to "runtime violation".	
US 3-015		4.1.1	8	te	The hardened implementation should be specified independently of contracts which are not a proven technology. All existing deployed field experience are not based on contracts.	Change "contract violation" to "runtime violation".	
RU-016		4.1.1	General [intro.compliance.general] p8	ge	Hardening and contracts are long awaited and desired safety features. We're not expecting that those features would decrease interest in "profiles" feature and we are eager to see "profiles" in C++29 or even soon after the C++26.	Keep the standard library hardening and the contracts as a way to customize it	
US 4-017		4.1.2	5	te	Describing the observable behavior of programs with undefined behavior is significantly complicated by the possibility of losing volatile operations. This distinction from C23 necessitates a new, obscure library function (std::observable_checkpoint) that is specific to the volatile case; there is moreover no evidence that it would actually change the behavior of any implementation. SG23 strongly recommended that something equivalent to the C23 rule be adopted, and EWG failed to reach consensus for that direction by the narrowest of margins.	Confirm with users of volatile that there is no need for aggressive optimization (apparently beyond current techniques) of volatile operations preceding a potentially undefined operation; if so, give volatile operations the same status as library I/O functions and simplify the specification accordingly (removing observable_checkpoint in particular).	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 5-018		5		te	Establish a definition for "whitespace character" as that term is used repeatedly and significantly in clauses 5 and 15.	Apply paper P3657r0.  p3657r0.pdf	
US 7-019		5.2		te	As we are updating the phases of translation in C++26, take advantage to split phase to convert preprocessor tokens to tokens as a distinct phase before translating the TU. This lines up better with modules and header units being an input into the phase after tokenization is complete.	Assuming phase 6 merges into phase 5 above, create a new phase 6 (preserving phase numbering above) to perform the pp-token to token conversion, prior to phase 7 starting with a full set of (converted) tokens.	
US 6-020		5.2		te	As we are updating the phases of translation in C++26, take advantage to merge phases 5 and 6 which deal with the same contiguous sequences of string literals.	Merge phase 5 and 6 of translation into just one phase.	
US 8-021		5.5	p1	te	The program is ill-formed if it has a preprocessing token matching the category of a single "other" character. [cpp.pre]p5 relaxes the directive syntax to ignore all preprocessor tokens within a skipped group. Surveying known implementations, all believe that [cpp.pre] dominates, so add a note/normative text to [lex.pptoken] to endorse the current behavior.	Revise in [lex.pptoken]p1: "If a U+0027 APOSTROPHE, a U+0022 QUOTATION MARK, or any character not in the basic character set matches the last category, the program is ill-formed unless skipping a group [cpp.cond]."	
CA-022		5.5	Paragraph 1	ed	The restriction If any character not in the basic character set matches the last category, the program is ill-formed. can benefit from an illustrative example that it reserves space for future evolution of the language and for potential "conforming extensions". For example (where <U+3000> represents U+3000 IDEOGRAPHIC SPACE): # if 0	Add the example or clearly document reaffirmation that the example code is ill-formed as a violation of the quoted diagnosable rule.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<pre>#<U+3000>else // ill-formed, diagnostic required int x; # endif</pre> <p>can, with a diagnostic, be accepted as expanding to</p> <pre>int x;</pre> <p>via an extension that treats Unicode whitespace as C++ whitespace.</p>		
US 10- 023		5.11, 11.01, 11.2, A.9, C.1.4, C.1.6		ge	<p>The new identifiers with special meaning "trivially_relocatable_if_eligible" and "replaceable_if_eligible" are embarrassingly verbose. They are too long for practical use.</p> <p>They will not see use in industry because their semantics do not match the semantics requested by over a dozen industry veterans representing seven major projects (Amadeus, Blender, Boost, Parlay, Folly, Qt, HPX) across P3233, P3236, P1144R13, P3780, etc.</p> <p>Compiler vendors will have to add a `__keyword` or `[[attribute]]` anyway, to permit their library vendor to optimize types like `unique_ptr` in pre-C++26 modes. Library vendors cannot use the new identifiers with special meaning outside of C++26 mode.</p> <p>Even in C++26 mode, a compiler-specific "opt-in" (P1144-style) attribute or keyword may be needed in order to support constexpr `optional` and `inplace_vector`, for which the P2786-style warrant is insufficient.</p> <p>The new identifiers as such are useful only for user-defined resource-management classes, in C++26-only codebases. (Rule-of-Zero classes do</p>	<p>Adopt P3823R0 "Wording for US NB comment 10".</p> <p>Alternatively, adopting all of P1144R13 "std::is_trivially_relocatable" would moot this issue.</p> <div>   </div> <p>p3823r0.html p1144r13.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)






² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>not need them. C++23 codebases must use a vendor-specific mechanism such as an attribute.) This is a very small fraction of a small fraction: permitting this syntax gains very little in performance, but loses much in confusion to the industry and embarrassment to the community.</p> <p>For all these reasons these identifiers should be removed from C++26, and their design reconsidered for C++29 after there has been any implementation and usage experience. We can safely remove these identifiers with special meaning, without at all harming the library feature of "relocation" itself.</p> <p> p3233r0.html  p3236r1.html</p> <p> p1144r13.html  p3780r0.html</p> <p> p2786r13.html</p>		
US 9-024		5.11, 6.09.1, 7.5.6.2, 11.1, 11.2, 16.4.6.11, 21.3, 21.4, A.9, C.1.4, C.1.6		te	<p>The new notion of "replaceable types" (6.9.1) is confusingly similar to the existing notions of "transparently replaceable objects" (6.8.4) and "replaceable functions", such as replaceable allocation functions (9.6.5) and replaceable contract-violation handlers (6.11.3).</p> <p>The new notion of "replaceable" is not used by any C++26 library machinery. Therefore it does not need to exist in C++26. The proposed change is entirely removal of various terms of art all of</p>	<p>Adopt P3827R0 "Wording for US NB comment 9", which proposes the following edits only:</p> <p>In 5.11: Strike "replaceable_if_eligible" from Table 4.</p> <p>In 6.9.1: Strike the sentence "Cv-unqualified scalar types, replaceable class types (11.2), and arrays of such types are collectively called replaceable types."</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>which are dangling loose ends. The removal of this type trait has no effect on the rest of C++.</p> <p>There are 88 instances of the string "replaceab" in the Committee Draft. Adopting the proposed change will remove</p> <ul style="list-style-type: none"> - 36 instances concerned with "replaceable_if_eligible" <p>leaving:</p> <ul style="list-style-type: none"> - instances concerned with "system_context_replaceability" - 4 concerned with "transparently replaceable" - 35 instances concerned with replaceable functions 	<p>In 7.5.6.2: Strike "whether the closure type is replaceable (11.2), or"</p> <p>In 11.1: Strike "replaceable_if_eligible" from the grammar. Replace the words "trivially_relocatable_if_eligible, or replaceable_if_eligible" with "or trivially_relocatable_if_eligible".</p> <p>Strike 11.2/6 and 11.2/7 in their entirety. Strike "or replaceability" from Note 2 and "or replaceable" from Note 3.</p> <p>Strike 16.4.6.11/3 in its entirety.</p> <p>In 21.3.3: Strike "is_replaceable" and "is_replaceable_v".</p> <p>In 21.3.6.4: Strike "is_replaceable" from Table 54.</p> <p>In 21.4.1: Strike "is_replaceable_type".</p> <p>In 21.4.17: Strike "is_replaceable_type".</p> <p>In A.9: Strike "replaceable_if_eligible".</p> <p>In C.1.4: Strike the words "and replaceable_if_eligible". Strike the words "and replaceable". In Example 1, replace "replaceable_if_eligible" with "trivially_relocatable_if_eligible".</p> <p>In C.1.6: Strike the words "or replaceable_if_eligible".</p> <p>Alternatively, adopting all of P1144R13 "std::is_trivially_relocatable" would moot this issue.</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						 p3827r0.html	
FR-002-025		5.12	Table 5	te	The class property specifiers <code>trivially_relocatable_if_eligible</code> and <code>replaceable_if_eligible</code> are not keywords to avoid clashing with user-defined identifiers. However, that makes it impossible to extend them into a conditional version in future C++ versions, similar to how <code>explicit</code> can receive a boolean condition. Given that the identifiers are quite verbose, making them keywords is unlikely to break anything.	Make <code>trivially_relocatable_if_eligible</code> and <code>replaceable_if_eligible</code> keywords by adding them to Table 5.	
US 11-400		6–14		ge	The document confuses the plain text term "declaration" with the grammar production declaration in several places. Perform a thorough review of each usage of the term to confirm it is rendered in the correct style. Uses as a compound noun can be safely ignored, e.g., function declaration, template declaration.	Perform a thorough review of each usage of the term "declaration" to confirm that it is rendered in the correct style.	
US 12-026		6.2	p2	te	This paragraph refers to declarations according to the grammar term defined in clause 9, but multiple entries in the list are not subject to that grammar, but rather of the more broadly defined plain-text term grammar defined in the preceding [basic.pre]	Change the font for "declaration" from a grammar term to plain text.	
US 13-027		6.4.1	1	te	The list of scope introducers is incomplete.	Add lambda expressions.	
CA-028		6.4.6	Paragraph 1	te	As documented by Core Issue 3033, "Scope after <i>declarator-id</i> before determining correspondence", the specified scope may be unknown when	Consider the approach documented in CWG 3033.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					knowledge of it is required. Such a requirement may come from a call to <code>access_context::current()</code> .		
US 14- 029		6.5.1	p2	te	There is no clear definition of a program point, which is fundamental to a lot of the syntax rules in many clauses. This reference in name lookup seems the closest to a definition, but we might prefer an entry in clause 3?	Define a program point such that, in translation phase 7, "There is a program point between every pair of adjacent tokens, before the first token of the translation unit, and after the last token of the translation unit."	
US 17- 030		6.7	10	te	All C++ implementations have converged on "strong ownership" of modules, with none planning to implement "weak ownership". This disposition no longer serves any purpose other than removing certainty.	Codify only the strong ownership model as follows: "If two declarations of an entity are attached to different modules, the program is ill-formed if one declaration is reachable from the other."	
FR- 003 -031		6.7	10	te	All C++ implementations have converged on "strong ownership" of modules, with none planning to implement "weak ownership". This disposition no longer serves any purpose other than removing certainty.	Codify only the strong ownership model as follows: "If two declarations of an entity are attached to different modules, the program is ill-formed if one declaration is reachable from the other."	
US 15- 032		6.7	p1	te	A program is defined as one or more translation units linked together, but translation units are defined as just a sequence of tokens and linking (translation phase 8) takes translated translation units as its input.	Update p1: A program consists of one or more <code><ins>translated</ins></code> translation units (5.2) linked together.	
US 16- 033		6.7	p4	te	Two different definitions for the term "attach to a module": <code>[basic.link]p4</code> and <code>[module.unit]p7</code>	Pick one as the primary definition. Then (editorially) cross-reference uses of the term attach to that definition.	
BDS2 -034		6.8.1 [basic.types.general], 11.2 [class.pro		te	In the current draft, trivial relocation can only be performed by the standard library function <code>std::trivially_relocate</code> , which requires that the type of the objects in the storage region be known. This is too restrictive and does not correspond to how current C++ libraries implement relocation.	Change the definition of trivially relocatable in <code>[basic.types.general]</code> and <code>[class.prop]</code> such that a trivially relocatable type can be relocated via <code>memcpy</code> or equivalent. This makes "trivially relocatable" exactly analogous to the existing "trivially copyable"	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
		p], 20.2.6 [obj.lifeti me]			The existing practice is that objects are relocated via copying the storage bytes, using memmove or equivalent. This is much more useful in practice, because it allows the use of realloc, allows non-C++ languages (such as C code, f.ex. OS kernel code) to move blocks of memory even if they contain trivially relocatable C++ objects, and allows C++ code to relocate blocks of memory without knowing the types of the contained C++ objects.	concept, in that both refer to the use of memcpy to relocate or copy an object. Note that this concept already exists in a hidden form; whether union { T t; } is trivially relocatable (by the current draft definition) is exactly whether T can be relocated by copying the untyped bytes (i.e. T is trivially relocatable by the new definition). However, since the core language does not at present guarantee anything about trivial relocation using memcpy or equivalent, C++ code can't take advantage of the above fact.	
US 18- 035		6.8.4	8	ed	The comma in "or, after" is spurious.	Remove it.	
GB02 -036		6.8.5	2	te	Fix erroneous behaviour termination semantics With the current specification of erroneous behaviour (EB), a program effectively enters an "erroneous state" once any erroneous behaviour happens, and may be terminated at any arbitrary point in time after that. The expectation is that it will be either reasonably soon or never, but the specification gives significantly more implementation freedom. This degree of freedom makes it much harder to reason about the behaviour of the program. It also presents a significant impediment to applying the concept of EB more broadly across the C++ language to remove undefined behaviour (in particular in cases unrelated to uninitialised values). These issues can be addressed without losing compatibility with existing implementations by removing the "delayed termination" aspect from the specification and instead making erroneous	Apply the changes proposed in P3684R0. Remove the "delayed termination" aspect from the specification and instead make erroneous values propagate out of operations that produce them.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					values propagate out of operations that produce them.		
US 19- 037		6.8.7	5	ed	The change to use “several” instead of a count loses the implication that the contexts listed below are exhaustive.	Clarify that lifetime extension is not otherwise performed.	
US 21- 038		6.8.7	7,8	te	This specification has numerous issues. For example, an enumerating expansion statement does not have a <i>for-range-initializer</i> or a <i>full-expression</i> to speak of. To the extent that this is intended for the range-for analog, that’s an iterating expansion statement (which also does not have a <i>for-range-initializer</i>).	Resolve Core issue 3043.	
US 20- 039		6.8.7	p7	te	P1306R5 added a lifetime extension rule for enumerating expansion statements, which was subsequently discovered to be incorrectly worded. This is CWG3043. However, the “Possible resolution” listed for that issue would remove the lifetime extension rule entirely, causing temporaries created within a given element of the braced list to be destroyed before executing any other code associated with the corresponding iteration. Such a specification is likely to cause bugs similar to those that were fixed by the lifetime extension rule for range-based for loops (adopted in C++23).	Amend the proposed resolution to CWG3043 so that the lifetime of temporaries created within each element of the expression-list is extended to that of the variable created by the for-range-declaration, then resolve CWG3043 by adopting the resolution.	
CA- 040		6.8.7	Paragraph 6	te	P2748R5, “Disallow Binding a Returned Glvalue to a Temporary”, created a misfeature that introduces lifetime extension of temporaries created during the evaluation of a function to opaque lifetimes past function return.	Resolve Core Issue 3063, “Lifetime extension of temporaries past function return”.	
CA- 041		6.8.7	Paragraph 6	te	The resolution of Core Issue 2894 was applied without updating the set of casts that lifetime extension for reference binding “looks through”, resulting in Core Issue 2941, “Lifetime extension for function-style cast to reference type”.	Apply the suggested resolution in CWG 2941.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 22- 042		6.9.2	17	ed	The list in paragraph 17 contains 21 items but example 2 illustrates only 18 of them.	Add examples illustrating the missing items (function parameter, annotation, and alias template).	
US 23- 043		6.10.2.3	5	ed	Note 4 is incorrect with the addition of bullets 1.2 and 1.6.	Strike the note.	
US 24- 044		6.10.3.4	3	te	It is not clear whether "completion of the constructor" includes delegating constructors.	Specify that only non-delegating constructors (for the most derived type) are considered.	
CA- 045		6.10.3.4	Paragraph 3	te	<p>The subject paragraph conflates the idea of completing a constructor (which, in theory, can include the non-delegating constructor called by a delegating constructor) and completing dynamic initialization of an object (possibly including the destruction of any temporaries). Additional clarity over what happens for block-scope statics initialized as part of executing destructor calls for temporaries of the initialization construct is needed.</p> <p>Also, the subject paragraph talks about the destructor of objects (of which arrays of class objects have none) and is incompatible with the treatment (consistent with the 2024-11-22 proposed resolution to Core Issue 2929, "Lifetime of trivially-destructible static or thread-local objects") where cleanup registration does not happen individually for the destruction of array elements.</p> <p>Furthermore, the word "during" in the last sentence of the subject paragraph seems imprecise in the presence of asynchronous execution.</p>	<p>Adopt the proposed resolution to Core Issue 2929.</p> <p>Also, modify 9.5.1 [dcl.init.general] paragraph 22 as follows:</p> <p style="padding-left: 40px;">An object whose initialization has completed (<u>including, if the initialization is a full-expression, the destruction of associated temporaries</u>) is deemed to be constructed, even if the object is of non-class type or no constructor of the object's class is invoked for the initialization.</p> <p>Additionally, modify 6.10.3.4 [basic.start.term] paragraph 3 as follows:</p> <p style="padding-left: 40px;">If the completion of the constructor or dynamic initialization deemed construction of an a complete object with static storage duration strongly happens before that of another, the completion of the</p>	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>Additionally, without changes like the proposed resolution to Core Issue 2929, that last sentence raises questions as to whether the destruction of subobjects occurs when `exit` is called during initialization of a complete class object of static storage duration.</p>	<p>destructor <u>destruction</u> of the second is sequenced before the initiation of the destructor <u>destruction</u> of the first. If the completion of the constructor or dynamic initialization <u>deemed construction of an a complete</u> object with thread storage duration is sequenced before that of another, the completion of the destructor <u>destruction</u> of the second is sequenced before the initiation of the destructor <u>destruction</u> of the first. If an object is initialized statically, the object is destroyed in the same order as if the object was dynamically initialized. For an object of array or class type, all subobjects of that object are destroyed before any block variable with static storage duration initialized during the construction of the subobjects is destroyed.</p> <p>Append an example to 6.10.3.4 [basic.start.term] paragraph 3:</p> <p>In the following program, the elements of a are destroyed, followed by dt, and finally the two BTemp objects:</p> <pre> struct DTemp { ~DTemp(); }; struct Temp { ~Temp() { static DTemp dt; } }; </pre>	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02	Document:	Project:
-----------------	-----------	----------

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

						<pre>struct BTemp { ~BTemp(); }; struct A { const BTemp &tb; ~A(); }; A a[] = {(Temp(), BTemp()), BTemp()}; int main() {}</pre> <p>Finally, make similar changes to 6.10.3.4 [basic.start.term] paragraph 5 as follows:</p> <p>If the completion of the initialization deemed construction of an a complete object with static storage duration strongly happens before a call to <code>std::atexit</code> (see <cstdlib>, 17.5 [support.start.term]), the call to the function passed to <code>std::atexit</code> is sequenced before the call to the destructor for initiation of the destruction of the object. If a call to <code>std::atexit</code> strongly happens before the completion of the initialization deemed construction of an a complete object with static storage duration, the call to the destructor for completion of the destruction of the object is sequenced before the call to the function passed to <code>std::atexit</code>. If a call to <code>std::atexit</code> strongly happens before another call to <code>std::atexit</code>, the call to the function passed to the second</p>	
--	--	--	--	--	--	---	--

1 MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						<code>std::atexit</code> call is sequenced before the call to the function passed to the first <code>std::atexit</code> call.	
ES-046		6.11		te	In paper P3829 [examples are given on how contract assertions may lead to situations where a critical check may end-up elided. As the papers states this might be a powerful tool for new supply-chain attacks. Please see paper P3851 and P3829 for details.	This is another major safety issue. Contracts should not worsen the safety dimension of the language. Safety must be given the highest priority for C++26.	
ES-047		6.11		te	When there are multiple dependent assertions (a precondition that checks for null pointer and another precondition that dereferences that pointer), the evaluation in non-terminating modes may lead to undefined behavior. This is a safety issue. It may be mitigated by not evaluating dependent contract assertions when the first assertion fails. Please, see paper P3851 for details.	Provide a solution for multiple dependent assertions in non-terminating modes. Safety must be given the highest priority for C++26.	
ES-048		6.11		te	In P3835 examples are given for a header file that contains an inline function with a contract assertion and is used from different translations units with different evaluation semantics. The same applies to constexpr functions, consteval functions and templates. This a major safety issue as the same assertion might be checked or not depending on the caller. Yet subsequent code might depend on the validity of that assertion.	Provide a solution for mixed mode builds that is not safety concern. Safety must be given the highest priority for C++26.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					Please, see papers P3851 and P3835 for details.		
ES-049		6.11		te	<p>The contracts feature is a major feature that lacks of enough deployment experience. While it is true that some portions have been experimented in projects. There are many features that have not been tried sufficiently.</p> <p>Moreover, at this point we also lack of enough user experience, implementation experience and build system experience. The latter is specially significant in presence of mixed mode builds. We should also have experience in multiple domains. What is acceptable in one problem domain becomes critically unacceptable in a different domain.</p> <p>Please, see paper P3851 for details.</p>	The safest path would be to get more experience by providing the feature either in a technical specification or in a white paper, so that all the issues are better understood.	
ES-050		6.11		te	<p>Contract assertions in its current form exhibit several serious problems that should be addressed before incorporation into an international standard.</p> <p>Please, see paper P3851 for details.</p>	Either the concerns are addressed or the feature should be eliminated for the proposed standard .	
US 26-051		6.11		te	<p>The Contracts feature as specified has too many important implementation defined semantics, and does not have enough in-field deployment experience. It shall be removed.</p>	<p>All edits applied by P2900R13 shall be reverted.</p>  <p>p2900r13.pdf</p>	
US 25-052		6.11		te	<p>Contracts are not ready for standardization as specified in P3829R0 and P3835R0.</p>  <p>p3829r0.pdf</p>  <p>p3835r0.html</p>	<p>Contracts should be removed from the C++26 working draft until the safety and other issues can be resolved.</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FR-004 -053		6.11		ge	C++ Contract design Is insufficiently tested and will require modification as experience accumulates; that's not adapted for an international standard.	Contracts should be removed from the C++26 working draft until significant user experience of the proposed contract design has been accumulated. Experience with different contract designs is not sufficient. These contracts should be included specifically in a technical specification related to C++26. France would keen to support the elaboration of a new technical specification on that matter.	
FR-005 -054		6.11		te	All the paragraphs modified or added by P2900R13. The Contracts feature as specified has too many important implementation defined semantics, and does not have enough in-field deployment experience. It shall be removed.	All edits applied by P2900R13 shall be reverted.	
ES-055		6.11.1	4	te	Clause 6.11.1/4 makes any variable to be const within the predicate of a contract assertion. This also applies to the this pointer. This is specially problematic when invoking an overloaded function, as the overload resolution mechanism might select a different version than in other context. This is major concern from the teachability point of view, as it will make the code harder to understand. Moreover, this might not be acceptable for projects where maintainability and simplicity are major drivers. Please see paper P3851 for details.	Change the wording to avoid making variables const. Provide an alternate solution that does not change overload resolution	
RO 2-056	1-13	6.11.2		te	Although contract assertions ([basic.contract]) were introduced to improve the “safety and correctness of software” (as stated in https://wg21.link/P2900R14), the current design lacks adequate control and tunability, and relies	Append the functionality described in https://wg21.link/P3400R1 - Controlling Contract-Assertion Properties to the contract assertions feature.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>excessively on implementation-defined behaviour. This significantly limits its applicability in large codebases.</p> <p>In particular, the feature does not allow a function to be made “safe” in the sense of being provably correct at a given point in time and protected by contract assertions. The global contract semantics can be altered at any time, rendering those assertions effectively no-ops.</p> <p>In large codebases, this undermines reliability and consistency, and in practice will result in the feature being disabled or forbidden, thereby defeating its intended purpose.</p>	<p>This would address the above issues by providing fine-grained control over contract behaviour.</p> <p>Alternative resolutions:</p> <ul style="list-style-type: none"> Remove the ignore semantic for C++26. This could be added later, together with the properties feature. This would impact lib hardening, which needs another way to be configured - that can be addressed in a separate topic. Remove the Contract assertions feature entirely from C++26 and continue development for possible inclusion in C++29. 	
AT1-057		6.11.3		te	There is no programmatic way to check for the implementation-defined replaceability of the <i>contract-violation handler</i> . Providing one if not supported is IFNDR.	Add a feature test macro to <code>version.syn</code> (17.3.2) to query whether the contract-violation handler is replaceable (analogous to <code>__cpp_lib_freestanding_operator_new</code>).	
CZ4-058		7.5.5.2	items 6.1, 6.2 and 7.3	te	Contracts make referenced outside variable const which is to avoid common error when a modification in an <code>assert(...)</code> leads to conditionally evaluated code based on compilation mode. This is useful, but it makes the <code>contract_assert(<EXPR>)</code> represent asserting of other expression then visually is obvious, mostly because such operations will select different code path (overloading, template instantiation) This makes automatic assertion insertion by tooling harder (they need to introduce <code>const_cast</code> on every external variable to remove the const-ness and keep the asserted expression meaning exactly same).	Remove the constification (<code>const T => T</code> on 6.1, <code>const U => U</code> on 6.2 and <code>const T => T</code> on 7.2) and replace any attempt to modify referenced outside objects / values erroneous behavior. This will keep the intent of constification from P2900, but won't introduce a new asymmetry into the language.	
BDS3-004		7.5.9	[<code>expr.prim.splice</code>]	te	Currently, splicing a constructor is explicitly disallowed. This is unnecessarily restrictive.	Remove “if S a constructor” from the ill-formed bullet in [<code>expr.prim.splice</code>], and add a bullet such that splicing when S is a constructor denotes a synthesized free function with the same	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						<p>parameters as S and with a return value of the corresponding type.</p> <p>This allows the form</p> <p>[: ct :](1, 2)</p> <p>and the form</p> <p>&[: ct :]</p> <p>which are both highly useful in practice.</p>	
US 27- 059		7.5.9 7.6.1.5	2 6	te	7.5.9 makes no provision for splicing a reflection representing a direct base class relationship, yet 7.6.1.5 presupposes that this is possible.	Add specification for splicing such a reflection to 7.5.9. Update the description of value categories in 7.2.1 to include such an expression. Disallow the use of such expressions outside of a class member access (similar to the existing rule for non-static data members).	
US 28- 060		7.5.9	4	te	Splicing a concept is disallowed but splicing a variable template is allowed, which is inconsistent. Any issues related to dependent concepts should have already been resolved with the addition of concept template parameters.	Allow splicing concepts.	
US 30- 061		7.6.1.10	6	te	Both the note and the following normative rule are outdated by the allowance for noexcept to be missing from the apparent type.	Specify that the pointer value is unchanged (as for any reinterpret_cast between unrelated object types) and update the note to use "call-compatible".	
US 29- 062		7.6.1.5	2	te	If the second expression represents a direct base class relationship, the first expression should be required to be a glvalue.	Add this requirement.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 32- 063		7.7		te	<p>Many reflection APIs either return vectors which require allocation (e.g. members_of) or deliberately avoid returning objects which require allocation (e.g. identifier_of). Using vector is convenient for many uses, but it pushes the issue front and center onto all reflection users that we do not yet have non-transient allocation. While workarounds exist, they are cumbersome and verbose at best.</p> <p>While we do not have a good, general solution to this problem, a significant amount of value can be derived from taking the two most common allocating types (std::vector and std::basic_string) and simply blessing those allocations as being acceptable. This would additionally allow reflection APIs to return std::string instead of a null-terminated std::string_view, which is simply bizarre.</p>	<p>Adopt P3554.</p>  <p>p3554r0.html</p>	
US 31- 064		7.7		te	<p>We currently do not have a way to represent values that can only exist at compile-time, and the lack of ability to do so means that we don't have a satisfactory way to implement variant visitation when consteval functions are involved (LWG 4197). Additionally, while constexpr up until now always means runtime or compile-time, it can now also mean compile-time-only depending on the contents of the variable. This is a messy model, which brings with it complexity but cannot actually solve the problems it needs to solve.</p> <p>The introduction of consteval variables — as properly compile-time only variables — allows for a very clean solution to the variant visitation problem as well as a clean model for variables in general: we can allow constexpr to continue to always mean runtime or compile time while consteval in all contexts means compile-time-only.</p>	<p>Adopt P3603.</p>  <p>p3603r0.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 33- 065		7.7	1	te	It is not clear when constant evaluation actually takes place, as is becoming increasingly important given new and upcoming features like reflection and constexpr output.	Specify when during translation and/or execution constant initialization and destruction (including of local variables) and the evaluation of manifestly constant evaluated expressions take place.	
US 35- 066		8.1	8	te	The second sentence is redundant to the first as class and enum definitions can only be defined by a <i>defining-type-specifier</i> .	Strike it.	
US 34- 067		8.1	8	te	This restriction should apply to the <i>structured-binding-declaration</i> of the <i>for-range-declaration</i> too.	Add “or the <i>for-range-declaration</i> ” after “of the <i>condition</i> ”.	
DE- 068		8.7 [stmt.expand]		te	Expansion statements have several inconsistencies, as discussed in core issues 3043, 3044, 3045, and 3048.	Apply the suggested resolution of the aforementioned core issues.	
US 36- 069		8.8.3	1	te	There is no <i>iteration-statement</i> production that contains a <i>compound-statement</i> .	Strike “or <i>compound-statement</i> ”.	
US 37- 070		9.1	1	ed	The plain text term declaration has a distinct meaning from the grammar production declaration and the two are easily confused. Rename one to remove confusion and to make misuse stand out more.	Rename the grammar production declaration to namespace-scope-declaration.	
FI- 071		9.4		te	The contracts facility has four major problems: - lack of sufficient implementation experience - lack of sufficient deployment experience - the standard gives a specific and terse syntax	Remove the contracts facility from C++26 and ship the current form of it in	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>for a contract-checking facility that is not a safety facility.</p> <p>- we in particular have no implementation or deployment experience on non-Itanium ABIs and Microsoft is telling us that they don't think it's feasible to have exceptions thrown from contract predicates translated into contract violations. This is a significant component of a situation where the facility as standardized won't be portably available, and when it's not portably available, it's not useful to standardize it.</p> <p>All of these parts result in standardizing an immature and unbaked and untested-by-users facility that we will regret in the forthcoming years, and it'll make it more difficult to adopt a facility that would strive for being an actual safety facility, especially one that would be safe by default, and a facility that would provide serious support for tools like static analysis tools.</p>	a White Paper or a Technical Specification instead.	
ES-072		9.4.1	6	te	C++ supports multiple styles of programming, including object-oriented programming where virtual functions play a major role. However, the contract feature explicitly forbids the use of	Provide a solution to support pre/post-conditions in virtual functions.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					contracts specifiers for virtual functions (clause 9.4.1/6). This is a major drawback of the feature. There are large codebases written in C++ that make extensive use of virtual functions. By providing a solution that does not support one of the major styles the language is doing a disservice to such portion of the user community. Please see paper P3851 for details.		
ES-073		9.4.1	7	te	Clause 9.4.1/7 states that by a using a non-reference parameter of a function in a post-condition the parameter magically becomes const. In this case, adding a post-condition to a function is changing the signature of the function, which seems counter-intuitive. This is major concern from the teachability point of view, as it will make the code harder to understand. Moreover, this might not be acceptable for projects where maintainability and simplicity are major drivers. It is also worth to note that constification violates a fundamental design principle for C++: given the same object, the same operation should have the same effect. Please see paper P3851 for details.	Avoid constification of function parameters because of mentioning them in a postcondition.	
ES-074		9.4.1	8	te	Clause 9.4.1/8 states (inside a note) that “ A pointer to function, pointer to member function, or function type alias cannot have a function-contract-specifier-seq associated directly with it”. Pointer to functions are a first class citizen in C++ that should not be neglected. Please see paper P3851 for details.	Provide a solution to support pre/post-conditions in pointer to functions, pointer to member functions, and function type aliases.	
US 38-075		9.5		te	It is still not possible to use designated-initialization to properly initialize aggregates with base classes, despite aggregates being able to have base classes since C++20. This is a defect that needs to be resolve.	Adopt P2287R5, which is also already basically done with Core review.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						 p2287r5.html	
US 39- 076		9.5.1	8	te	A class that contains only std::meta::info data members, or arrays thereof, should be const-default-constructible.	Make the definition of const-default-constructible properly recursive instead of having a special case for std::meta::info.	
US 40- 077		9.5.5 List-initialization [dcl.init.list]	6	te	<p>“The backing array has the same lifetime as any other temporary object (6.8.7), except that initializing an initializer_list object from the array extends the lifetime of the array exactly like binding a reference to a temporary.”</p> <p>Except that it is not exactly the same because when the lifetime of a temporary is extended it still has automatic storage duration changed from the statement to the block, like a declared variable. What is proposed is different because it is being extended by being turned into something that has static storage direction. In a note, far from this standard verbiage, it is said to be permitted as in it may or may not happen and the decider is the compiler. This is unacceptable. Lifetimes must be clearly defined and the criteria for lifetimes need to be clearly delineated especially for static storage duration otherwise it would lead to unresolved use after free. If the programmer thinks it is static based on the provided example but the compiler left it as automatic storage duration, then the programmer likely did not make the necessary code changes to ensure it doesn't dangle. Not being able to trust this feature can cause programmers to not trust their compilers and to program around this by writing more esoteric legacy code. The lack of definiteness is not portable between compilers that target the same standard and even worse with just one compiler where the desired</p>	<p>Add verbiage to state static storage duration. The proposed objective was for static storage duration but the only mention of it is in an example and a foot note in comment US 268.</p> <p>https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2752r3.html</p>   p3824r0.html p2752r3.html	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02	Document:	Project:
-----------------	-----------	----------

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>behaviour could vary based on compiler flags or added and removed due to optimization. This breeds use after free unsafety. The verbiage should be as definite as it is for string literals that it DOES have static storage duration.</p> <p>5.13.5 String literals [lex.string] Paragraph 9</p> <p>“Evaluating a string-literal results in a string literal object with static storage duration (6.8.6).”</p> <p>Should a initializer list of ints or even chars be specified radically nebulous from the clear straight forward verbiage of string literals.</p> <p>A programmer can't be expected to be responsible for dangling, if the programmer can't definitely reason about the lifetimes of objects. A code reviewer/code auditor can't be expected to find dangling, if the reviewer/auditor can't definitely reason about the lifetimes of objects. None of these three people should be forced to look at assembly, a different programming language, to know what the compiler decided to do in this instant and worse to repeat the process for each initializer list/braced initializer in a program.</p> <p>Another issue that may or may not need to be fixed is the original proposal was for “Static storage for braced initializers” yet the current verbiage is only for initializer lists. If the currently worded functionality is reasonable even for an initializer list of size one, why not for single instance objects that are braced initialized entirely of constant literals. Might this be viewed as an inconsistency and even worse, a missed opportunity to fix a swath of use after free by lifetime extending objects to have static storage duration.</p>		
US 42- 078		9.13.12	1	ed	Declaration of function parameters are not explicitly spelled out as being a valid target.	Add "or to a parameter-declaration of a function declaration"	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 41- 079		9.13.12	1	te	base-specifiers are not declared and so do not have declarations.	Reword the sentence.	
US 43- 080		10.2	7	te	The note about linkage is not correct for header units.	Mention their internal-linkage case.	
US 45- 081		11.1		te	<p>The trivial relocation language feature exists to allow implementations to memcpy data instead of performing a move followed by a destroy.</p> <p>Approximately everyone learning about trivial relocation will hear it described in this way, and approximately everyone in the C++ community will assume as a result that memcpying data <i>is well defined</i>, especially since that is what existing practice looks like. That is, that for any trivially relocatable type T, the following is well-formed code:</p> <pre>auto make() -> T; auto consume(T const&) -> void; auto f() -> void { // put a T in buffer alignas(T) std::array<unsigned char, sizeof(T)> buffer; new (buffer) T(make()); // perform a memcpy alignas(T) std::array<unsigned char, sizeof(T)> buffer2 = buffer;</pre>	Either the memory model should be adjusted so that memcpying trivially relocatable types is well-defined behavior, or the feature should be removed until we can do it correctly.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:


MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<pre>// expect a T in buffer 2 consume(*reinterpret_cast<T const*>(buffer2.data())); }</pre> <p>However, this code is undefined if T isn't trivially copyable. Approximately nobody will get this right. Although it may just happen to work.</p>		
US 44- 082		11.1		te	<p>The trivial relocation language feature currently does not allow for the ability to mark a type as being <i>conditionally</i> trivially relocatable. The suggested workaround in the paper is that users manually have to add conditionally-non-relocatable base classes or members to achieve this.</p> <p>That workaround is atrocious, and means that we're adding a language feature in C++26 that has worse ergonomics than a library feature that we could implement in C++26. That's not an acceptable situation to be in.</p>	<p>In decreasing order of preference:</p> <ol style="list-style-type: none"> 1. The trivial relocation keywords should become attributes and accept a parenthesized conditional. 2. The trivial relocation keywords should become annotations that could be constructed or invoked with a bool. 3. The trivial relocation feature should be removed entirely. 4. The trivial relocation keywords should become full keywords and accept a parenthesized conditional. 	
CA- 083		11.1	Paragraph 5	ed	The use of “the identifier” as the introduction for a list of multiple identifiers is ungrammatical.	Replace “the identifier” with “one of the identifiers”.	
US 47- 084		11.2	2	Te	Implicitly deleted move operation should not disable trivial relocation	Change each occurrence of “nor deleted” (3 places) in the bullet list of paragraph 2 to “nor explicitly deleted”, as per the suggested resolution to CWG issue 3049 .	
US 46- 085		11.2, 20.2.6		te	So-called "trivial" relocation is currently permitted to do arbitrarily more than a bitwise copy of the object representation. This has at least four bad effects:	In 11.2: Strike the words "except that it is implementation-defined whether an otherwise-eligible union having one or more subobjects of	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02	Document:	Project:
-----------------	-----------	----------

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>(1) It means that <code>std::is_trivially_relocatable</code> returns true for types that are not in fact "trivially relocatable" by the industry definition.</p> <p>Google Gemini, when asked "what is trivially relocatable type in c++", responds:</p> <p>> A trivially relocatable type in C++ is a type whose objects can be moved from one memory location to another by simply copying their raw bytes (e.g., using <code>memcpy</code>) without needing to call the move constructor at the new location and the destructor at the old location. This allows for significant performance optimizations [...] Key characteristics and implications of trivially relocatable types include:</p> <p>> Bitwise Copying: The core idea is that the object's internal state remains valid and consistent after a bitwise copy to a new address. [...]</p> <p>> Safety and Correctness: The concept ensures that relocating an object via a bitwise copy is a safe and correct operation, preserving the object's invariants and avoiding undefined behavior.</p> <p>It would be unfortunate if C++26 standardizes a meaning for "is_trivially_relocatable" that is inconsistent with this widely understood and well-supported definition. Programmers (and LLMs) *will* assume that trivially relocatable types can in fact be <code>memcpy</code>'ed, <code>realloc</code>'ed, <code>mmap</code>'ed with safety. C++ should provide them that safety, not undercut them.</p> <p>(2) "Trivial" means "bitwise" in every other situation (e.g. "trivially copy constructible" always means "as if by <code>memcpy</code>," never anything more</p>	<p>polymorphic class type is eligible for trivial relocation".</p> <p>In 20.2.6: Strike the words "except for any parts of the object representations used by the implementation to represent type information (6.8.2)".</p> <p>In 20.2.6 /10: Strike the precondition "No element in the range [first, last) is a potentially-overlapping subobject."</p> <p>In 20.2.6 /17: Strike the precondition "No element in the range [first, last) is a potentially-overlapping subobject."</p> <p>Alternatively, adopting all of P1144R13 "<code>std::is_trivially_relocatable</code>" would moot this issue.</p> <p> p1144r13.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>expensive: e.g. polymorphic types are never trivially copy constructible; ptrauth-qualified scalar types are never trivially copy constructible). Trying to make "trivial" mean something different from "bitwise" in the solitary case of relocation is inconsistent with the rest of C++.</p> <p>(2) It means that if you put two trivially relocatable types together in a union, the union itself might not be trivially relocatable. This makes trivial relocatability non-composable.</p> <p>(3) It introduces new implementation-defined behavior in 11.2, which could be made well-defined if we adopt the proposed change.</p> <p>(4) It introduces new undefined behavior in 20.2.6, which could be made well-defined if we adopt the proposed change.</p> <p>Namely, <code>`std::relocate(static_cast<Base*>(derivedSrc), static_cast<Base*>(derivedSrc)+1, baseDst)`</code> is well-formed on paper, but physically results in a "radioactive" baseDst object with the vptr of a Derived but the data members of a Base. 20.2.6/10 and /17 add preconditions (UB) on all functions that use relocation, in order to make sure the Standard doesn't accidentally claim that the "radioactive" behavior complies with the abstract machine. The proposed change makes <code>`std::trivially_relocate(static_cast<Base*>(derivedSrc), static_cast<Base*>(derivedSrc)+1, baseDst)`</code> ill-formed instead of UB, and makes <code>`std::relocate(static_cast<Base*>(derivedSrc), static_cast<Base*>(derivedSrc)+1, baseDst)`</code> well-defined instead of UB.</p>		

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
PL-003		11.4.5.2	[class.default.ctor]	te	<p>The changes to the [class.default.ctor] p4 from P3074R7 paper (starting lifetime of first implicit lifetime member), introduced a breaking change, when a type containing an union is used as NTPP.</p> <pre>union U { int a, b; }; template<U u> class X {}; constexpr U make() { U u; return u; } void f(X<make()>) {}</pre> <p>These changes were aimed to allow inplace_vector<T, N> to be more usable at compile time, but that functionality can be postponed.</p>	<p>Revert the following changes introduced by P3074R7:</p> <p>Additions to [class.default.ctor] p4:</p> <p>If a default constructor of a union-like class X is trivial, then for each union U that is either X or an anonymous union member of X, if the first variant member, if any, of U has implicit-lifetime type ([basic.types.general]), the default constructor of X begins the lifetime of that member if it is not the active member of its union. [Note 1: It is already the active member if U was value-initialized. — end note]</p> <p>Removal of [inplace.vector.overview] p4:</p> <p>For any N>0, if is_trivial_v<T> is false, then no inplace_vector<T, N> member functions are usable in constant expressions.</p> <p>Adding __cpp_lib_constexpr_inplace_vector to [version.syn].</p>	
US 48-086		11.4.5.2		te	<p>P3074 made unions trivial, but it did so in a way that breaks some (admittedly obscure) code but also doesn't help solving a closely related problem of wanting to have constexpr variables of types that have such unions inside of them. Both problems can be fixed by specifying the rules better.</p>	<p>Adopt P3726.</p>  <p>p3726r0.html</p>	
DE-087		11.4.5.2 [class.default.ctor], 11.5.1 [class.union.general], 7.7 [expr.const]		te	<p>Trivial unions in the CD are an essential improvement but the open CWG issue 2999 (Trivial unions changing existing behavior) should be addressed.</p>	<p>Adapt both proposal 1 and proposal 2 from P3726</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FR-006 -088		11.5.1		te	trivial unions (P3074) have been through many completely different design iterations. Yet, the version that has been adopted constitutes a breaking change (CWG2999). It is also harder to teach, or at least is less obvious than a library solution, while making the rules around unions even more convoluted.	Remove trivial union from C+26 and explore simpler library solutions(std::uninitialized) reflective of existing practices	
AT2-089		13.1 [temp.pre]	Paragraph 8.2	te	An entity defined in the for-range-declaration of an expansion-statement should also be templated. Currently, a structured binding pack can only be declared in the for-range-declaration of an expansion-statement if there is an enclosing template.	Change 8.2 to: an entity defined (6.2) or created (6.8.7) within the for-range-declaration or compound-statement of an expansion statement (8.7),	
US 49-090		13.2			Several parts of the standard and library reference structural types, including library mandates that types be structural, yet there is no way to query whether a type is structural. Note that the library mandates clauses mean that library implementers must somehow have this functionality, it is simply not exposed to users.	Add either a <code>is_structural</code> type trait, an <code>is_structural_type</code> meta function, or both.	
US 50-091		13.3	8, 9	ed	Paragraph 8 says that “names or designates” but paragraph 9 uses just “designates” when the subject and object are the same.	Make the paragraphs consistent using whatever the correct terminology is.	
CA-092		13.3	Paragraph 1	te	The change to the grammar of <i>template-argument</i> in P2841R7, “Concept and variable-template template-parameters”, causes dependent names used (without <code>template</code> qualification) as template arguments to be considered to name a non-type, non-template entity, causing any instantiation where the dependent name names a template to be ill-formed. This is seen in the treatment of the template argument in <code>g<T::TT></code> in the case of <code>A<Tmpl></code> in the example from Core	Apply the suggested resolution in CWG 3032 except that the last alternative should not use <i>constant-expression</i> (e.g., in case it is the argument to a template parameter of reference type). Instead, it should use just <i>expression</i> .	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					Issue 3032, "Template argument disambiguation". Said Core Issue also documents other similar issues in the area that can be addressed at the same time.		
FR-008 -093		13.5.2.5		te	A fold expanded constraint should have a parameter mapping behaving like the parameter mapping of an atomic constraint. Satisfaction of the Nth expansion of the fold expression substitutes the Nth corresponding argument from the expansion in the mapping.		
CA-094		13.5.5	Paragraph 1	te	The development of the "compatible for subsumption" restriction for ordering of constraints involving fold expressions was not sufficient to prevent the class of problems it was designed to handle. In particular, the effect of fold expressions in concept definitions was overlooked. This resulted in Core Issue 3021, "Subsumption rules for fold expanded constraints".	Consider the approach documented in CWG 3021.	
US 51-095		13.7.4		ed	In [temp.variadic], bullet (6.9) has two sentences: - In an attribute-list (9.13.1); the pattern is an attribute. In an annotation-list; the pattern is an annotation.	Each sentence in 13.7.4/(6.9) should be its own bullet point, like this: - In an attribute-list (9.13.1); the pattern is an attribute. - In an annotation-list (9.13.1); the pattern is an annotation.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 52- 096		13.7.8	2	ed	The first sentence uses “designates” but the second sentence uses “names”.	Make them consistent.	
FR- 007 -001		13.08		te	A number of issues introduced by P2893 (Variadic friends) persist which make part of the paper unimplementable	Adopt the proposed resolution to CWG2917	
US 53- 097		13.8.3.3	6	te	It is not specified when/if an <i>expansion-init-list</i> is type-dependent.	Add it to paragraph 6.	
AT3- 098		13.8.3.3 [temp.dep.e xpr]	Paragraph 3.6	te	This bullet was added by P1061R10 (Structured Bindings can introduce a Pack), but it also affects the type-dependence of non-structured binding packs (i.e., constant template parameter packs with non-dependent types).	Change 3.6 to “associated by name lookup with a structured binding pack”	
AT4- 099		13.8.3.4 [temp.dep.c onstexpr]	Paragraph 4	te	The special case for a structured binding pack introduces an unnecessary inconsistency. A structured binding pack with a non-dependent initializer should either always be instantiated early or never, regardless of the expression it is used in. The more comprehensive rule to always instantiate early was removed after R7 of the paper (P1061R7 Structured Bindings can introduce a Pack), but the special case for “sizeof ...” remained.	Delete “unless the identifier is a structured binding pack whose initializer is not dependent.”	
US 54- 100		13.10.3.1	8	te	The term “immediate context” is crucially important for understanding the behavior of templates, but has remained undefined in the Standard for over a decade. It is evident that a perfect definition is not forthcoming, but the Standard remains incomplete unless some definition, even an imperfect one, is provided.	CWG should determine a wording strategy for the resolution of CWG1844 (which may exclude unresolved design questions such as CWG2296) and solicit a paper to implement that direction to the greatest extent practicable.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
CA- 101		13.10.3.6	Paragraph 20	te	As documented in Core Issue 2900, “Deduction of non-type template arguments with placeholder types”, the wording has not been updated to handle template arguments (in either the P or A positions) declared using placeholder types.	Apply the suggested resolution in CWG 2900.	
US 55- 102		15.1		te	This prohibits all preprocessing directives before a module. This includes `#line` directives, which are used by implementations in various preprocessing or partial preprocessing modes. This currently breaks several different build systems that rely on these modes.	Allow `#line` and an implementation defined set of preprocessing directives at the start of the module-file grammar.	
US 56- 103		15.5		te	Paper P3034R1 made the declaration of private module fragments ill-formed in phase 4, as the pp-token private is an identifier in phase 4, so matches the grammar for a module partition without a module name.  p3034r1.html	Amend the grammar for pp-module, according to P(TBD).  p3838r0.pdf	
CA- 104		15.7.1	Paragraph 9	te	The implied behaviour that `likely` and `unlikely` can be defined as names of function-like macros but cannot be undefined does not seem to have strong rationale.	Extend the exception for `likely` and `unlikely` to include use with `#undef`.	
US 57- 105		15.7.1, C.1.6		te	[diff.cpp23.library]/2 currently says: Affected subclause: 16.4.6.3 [res.on.macro.definitions] Effect on original feature: Names of special identifiers may not be used as macro names. Valid C++ 2023 code that defines replaceable_if_eligible or trivially_relocatable_if_eligible as macros is invalid in this revision of C++.	In 15.7.1 /9, replace the phrase "the identifiers listed in Table 4" by "the identifiers with special meaning listed in Table 4". In C.1.6, replace "16.4.6.3 [res.on.macro.names]" by "15.7.1 [cpp.replace.general]". Add "identifiers with special meaning" as a term in the Index, pointing to [cpp.replace.general] and [lex.name] at least.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>"special identifiers" presumably means "identifiers with special meaning." But [res.on.macro.names] doesn't say that you can't define those as macros; in fact, [lex.name] strongly implies that you can. [macro.names]/2 used to say you can't, but that wording was recently moved to [cpp.replace.general]/9 instead:</p> <p>A translation unit shall not #define or #undef names lexically identical to keywords, to the identifiers listed in Table 4, or to the attribute-tokens described in [dcl.attr], except that the names likely and unlikely may be defined as function-like macros.</p> <p>And [res.on.macro.names] restricts macros defined by the Standard Library, not by user code.</p>	In 5.11 [lex.name], between sentences 2 and 3, insert: "[Note: Identifiers with special meaning are not permitted to be used as macro names [cpp.replace.general]. —end note]"	
US 58- 106		15.7.3	2	te	The order of evaluation between # and ## operators is unspecified, which means it is not only not portable between compilers, but has no guarantee to produce the same result on two successive translations of the same source file by the same compiler. As the user cannot take control of the ordering with parentheses, the order of evaluation should at least be implementation defined to bring consistency and predictability.	Replace unspecified in the last sentence with implementation defined. If all implementations produce the same definition, we might fully specify the order for the next standard.	
US 59- 107		15.7.4	3	te	The order of evaluation between two ## operators is unspecified, which means it is not only not portable between compilers, but has no guarantee to produce the same result on two successive translations of the same source file by the same compiler. As the user cannot take control of the ordering with parentheses, the	Replace unspecified in the last sentence with implementation defined. If all implementations produce the same definition, we might fully specify the order for the next standard.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					order of evaluation should at least be implementation defined to bring consistency and predictability.		
FR-009 -108		15.8		te	The restrictions in the range of values [1, 2147483647] does not match existing practice, some implementations accept 0 or a number greater than 2147483647	Make the value accepted by line a positive integer in an implementation-defined range,	
CA-109		15.12	Paragraph 1	te	The value of `__cplusplus` should be updated.	At the 2026-03 meeting, update the macro to 202603L	
US 60-110		15.13	1	te	There are multiple kinds of string literal prefixes now, not just the character L. C fixed their corresponding specification in C11.	remove the L and replace as "deleting the encoding-prefix, if present, ..."	
CA-111		15.13	Paragraph 1	te	The dstringization mechanism specified omits any treatment of the <i>d-char-sequence</i> , etc. of a raw string literal.	Make cases where the <i>string-literal</i> is a raw string literal ill-formed.	
US 61-112		16.3.2.4	03.4	te	The hardened implementation should be specified independently of contracts which are not a proven technology. All existing deployed field experience are not based on contracts.	Change "contract violation" to "runtime violation".	
FR-010 -113		16.3.2.4	03.4	te	The hardened implementation should be specified independently of contracts which are not a proven technology. All existing deployed field experience are not based on contracts.	Change "contract violation" to "runtime violation".	
US 62-114		16.3.3.2 16.3.3.7 16.4.2.5	1 1, 5 4	ed	The wording should be updated to reflect the fact that type aliases are now entities.	Remove the references to typedef-names from 16.3.3.2 paragraph 1, 16.3.3.7 paragraph 1, the introductory text of 16.3.3.7 paragraph 5, and 16.4.2.5 paragraph 4. 16.3.3.7 bullet 5.7 should now say "type alias" instead of "typedef-name".	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 63- 115		16.3.3.3.4.1	01.2	ed	Character sets should be defined in [lex.charset]. The execution character sets are referenced only in the definition of NTMBS in the library wording.	Move [character.seq.general]p(1.2) to immediately precede [lex.charset]p4.	
US 65- 116		17.3.2		te	There are forward declarations of entities from <spanstream> and <syncstream> in <iosfwd> so their feature macros should be added to that header too.	Add <iosfwd> to the "also in" entries for `__cpp_lib_char8_t`, `__cpp_lib_spanstream`, and `__cpp_lib_syncbuf`.	
US 66- 117		17.3.2, 23.7.3.2, 23.7.3.7.4, 23.7.3.7.5, 23.7.3.7.6.1 , 23.7.3.7.7		te	<p>As P3663 explains, it is currently undefined behavior if future C++ versions expand the set of valid slice specifier types for submdspan, and if the new types get used with a C++26 – compliant user customization of submdspan_mapping. Any addition to the tuple-like exposition-only concept would add to the set of valid slice types. This will happen in C++26 (std::complex is now tuple-like) and WG21 proposals in flight may make it happen in future C++ versions.</p> <p>P3663 fixes this by having submdspan “canonicalize” the open set of valid slice types to a fixed set of canonical types before calling the user’s submdspan_mapping, and by requiring that customizations of submdspan_mapping be ill-formed if called with non-canonical slice types. This makes behavior of C++26 – compliant customizations well-defined with new slice types.</p>	<p>Adopt the latest revision of P3663 (currently R2), “Future-proof submdspan_mapping.”</p>  <p>p3663r2.html</p>	
PL- 012		17.9	[support.exc eption]	te	Making current_exception/uncaught_exceptions constexpr, causes a breaking change, where the existing code changes its meaning for the	Address the breakage.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02	Document:	Project:
-----------------	-----------	----------

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>following code:</p> <pre>int const u = std::uncaught_exceptions(); bool const c = std::current_exception();</pre>		
US 67- 118		17.9.2, 17.9.6		te	<p>As pointed out on the CWG mailing list, both assertions here may fail in C++26:</p> <pre>const int n1 = std::uncaught_exceptions(); assert(n1 == std::uncaught_exceptions()); // FAIL int n2 = std::uncaught_exceptions(); assert(n1 == n2); // FAIL</pre> <p>Because `uncaught_exceptions()` became constexpr in C++26, the initializer for n1 can now be evaluated at compile-time, where it is invariably zero.</p> <p>C++23 was fine with a non-constexpr `uncaught_exceptions()`. We have discovered that constexpr `uncaught_exceptions()` causes real problems in the CD. The best course of action is to continue shipping C++23's non-constexpr `uncaught_exceptions()` for now; we will have the next three years to think about how (and whether) to constexpr-ify it.</p> <p>There is a similar issue with constexpr `std::current_exception()` and code like:</p> <pre>const auto ep1 = std::current_exception(); auto ep2 = std::current_exception(); assert(ep1 == ep2); // FAIL</pre> <p>Intuitively this latter issue seems less likely to bite programmers in practice, but perhaps it also should be fixed.</p>	<p>In 17.9.2, strike the word "constexpr" from constexpr int uncaught_exceptions() noexcept;</p> <p>In 17.9.6, strike the word "constexpr" from constexpr int uncaught_exceptions() noexcept;</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					In the long term, we suggest that C++29 should deprecate "trial constant-evaluation" of `const` scalar variables, and then C++32 remove trial constant-evaluation. Then n1's initializer would never be constant-evaluated, even if `uncaught_exceptions()` were made constexpr in C++32.		
GB03 -119		17.9.6		te	<p>Adding the <code>constexpr</code> specifier to <code>uncaught_exceptions()</code> and <code>current_exception()</code> causes breaking changes in trial constant evaluation</p> <p>Consider:</p> <pre>#include <exception> int i = 1; int main () { try { struct S { ~S() { // x is initialized to 0 in // trial constant evaluation const int x = std::uncaught_exceptions(); i = x; } }; S s; throw 0; } catch (...) { const bool x = (std::current_exception() != nullptr); return i + x; } }</pre> <p>The program returns 0, which is a breaking change compared to C++23, where it returns 2.</p>	Adopt the proposed wording change of P3842R0: "A conservative fix for <code>constexpr uncaught_exceptions()</code> and <code>current_exception()</code> ."	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>Previously discussed on the core and lib reflectors (same thread mirrored on both):</p> <p>https://lists.isocpp.org/core/2025/08/18460.php https://lists.isocpp.org/lib/2025/08/32521.php</p> <p>Discussed on LEWG telecon 9th Sep 2025: https://wiki.edg.com/bin/view/Wg21telecons2025/P3818#Library_Evolution_Telecon-2025-09-09</p>		
DE-120		<p>17.9.6 [support.exception.uncaught_exceptions],</p> <p>17.9.7 [support.exception.propagation]</p>		te	<p>Both std::uncaught_exceptions() and std::current_exception() refer to global state of the runtime, but are marked "constexpr".</p> <p>It is understood that the exception machinery during constant evaluation maintains state different from the runtime state, so it is unclear which state is referred to in situations where both could be in view, also in existing code, e.g.</p> <pre>const int x = std::uncaught_exceptions(); const bool b = std::current_exception() == nullptr;</pre>	<p>Remove "constexpr" from these two functions for C++26 and design a more comprehensive approach for C++29.</p>	
FI-121		17.9.6/17.9.7		te	<p>Making std::uncaught_exceptions() and std::current_exception() constexpr is a breaking change. Undo that breaking change, and do nothing else to these facilities. Figure out a better solution in the next standard.</p>	<p>Adopt the wording in P3842 that strikes the constexpr decl-specifier from the functions mentioned.</p>	
US 68-122		17.9.7	13-15	te	<p>template constexpr const E* exception_ptr_cast(const exception_ptr& p) noexcept; should be revised to use const</p>	<p>https://isocpp.org/files/papers/P3739R2.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					optional<&> instead of pointer as it is safer from null pointer dereference, pointer arithmetic and indexing issues	 p3739r2.html	
CZ3-123		17.9.7 [propagation]	current_exception and uncaught_exception	te	Interaction between constant evaluation and 7.7.8 [expr.const] “potentially constant” initialization is introducing a silent breaking change of code using const integer local variables to store result of uncaught_exception	Explicitly limit the breaking change by adding “Constant when: not evaluated within potentially-constant [expr.const] initialization.” as proposed in P3818R1	
GB04-124		17.10.3	5	te	Remove <code>evaluation_exception()</code> from contract-violation handling On some platforms, a conforming implementation of <code>std::contracts::contract_violation::evaluation_exception()</code> may have to copy the exception object thrown from a contract check before invoking the contract-violation handler. This copy may execute user code (copy constructor of a user-defined exception type). However, user code execution after a contract violation has been detected but before the contract-violation handler is called may pose a security risk. To mitigate any such risk, we should remove member function <code>evaluation_exception()</code> from the standard library type <code>std::contracts::contract_violation</code> . We can add this function back in future versions of C++ if it can be shown that the security risk is fully avoidable. In the meantime, the functionality it offers is available through other means.	Apply the changes proposed in P3819R0 Remove member function <code>evaluation_exception()</code> from the standard library type <code>std::contracts::contract_violation</code> .	
US 69-125		17.10.3	5	te	During implementation it was discovered that <code>evaluation_exception()</code> cannot be implemented with a trivial implementation in general --- something must be done to enable retrieving an <code>exception_ptr</code> that might not be the top of the exception stack when <code>evaluation_exception()</code> is invoked, and the exception facility does not natively provide that functionality. Given the tools	Adopt P3819R0 (i.e., Remove <code>contract_violation::evaluation_exception()</code>).  p3819r0.pdf	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>that an exception facility must currently provide an implementation must create an <code>exception_ptr</code> when an exception escapes a predicate, and that might involve copying the exception after a violation has been identified but before the violation handler is invoked (which would violate essential principles of Contracts in supporting the need to fail fast). See P3819R0 for additional discussion and rationale.</p>  <p>p3819r0.pdf</p>		
US 70- 126		17.12.7		te	<p><code>type_order</code> and <code>type_order_v</code> should allow incomplete types.</p>	Add explicit permission to do so.	
US 64- 127		17–33		ed	<p>Consistently remove every redundant typename after a using declaring a type alias in the standard library, as per P2150r0.</p>  <p>p2150r0.html</p>	<p>Adopt “3.2.1 Remove typename from Alias Definitions” from P2150r0.</p>  <p>p2150r0.html</p>	
US 71- 128		18.7.3, 24.3.4.5		ed	<p>In <code>[concept.regularinvocable]</code>, “This requirement supersedes the annotation in the definition of invocable” is confusing, because there are no annotations (9.13.12) in the definition of invocable.</p> <p>The same issue recurs in <code>[iterator.concept.inc]</code>.</p>	<p>In 18.7.3, replace the word “annotation” with “comment”.</p> <p>In 24.3.4.5, replace the word “annotations” with “comments”.</p>	
GB05 -129		19.3.3	3	te	<p><code>assert</code> should forbid uses of <code>co_await</code> and <code>co_yield</code></p> <p>Should not try to support things like <code>assert((co_await x, true))</code></p>	Add to paragraph 3: If its expansion contains the <code>co_await</code> or <code>co_yield</code>	

¹ MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						keywords, the program is ill-formed, no diagnostic required.	
US 72- 130		20		te	<p>P3516's std::uninitialized_relocate and std::uninitialized_relocate_backward were design-approved and entered LWG review, but are not actually present in the CD.</p> <p>These algorithms are convenient in order to implement user-defined containers similar to std::vector with relocation. They are used in practice already (6700 hits in a GitHub search for "uninitialized_relocate"). These algorithms are a flagship part of the "relocation" feature.</p> <p>We should not ship a "relocation" feature without these standard library algorithms.</p>  <p>p3516r2.html</p>	<p>Adopt P3516R2 "Uninitialized algorithms for relocation".</p>  <p>p3516r2.html</p>	
US 73- 131		20.2.2, 20.2.6		te	<p>std::relocate is redundant with the STL algorithms std::uninitialized_relocate and std::uninitialized_relocate_backward as design-approved in P3516.</p> <p>The latter algorithms are more general, and are used in practice (6700 hits in a GitHub search for "uninitialized_relocate"). The former function template is novel and unused in practice; the only Google hits for "std::relocate" are talking about a different function, from P1144R6, which has entirely different semantics from the CD's std::relocate. (P1144R6's "std::relocate" relocates from its argument into its return slot, enabling return-by-relocate. The CD's std::relocate simply</p>	<p>Adopt P3631R0 "Cleaning up the trivial relocation APIs in C++26".</p>  <p>p3631r0.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>duplicates the functionality of the CD's std::uninitialized_relocate.)</p> <div>   </div> <p>p3516r2.html p1144r6.html</p>		
RU-132		20.2.3.3	Member functions [pointer.traits.functions]	te	`pointer_traits::pointer_to` should be `constexpr` to implement some of the constexpr containers	Consider applying the remaining part of the fix from library issue 3454 and close it as fixed.	
CA-133		20.2.6		te	The std::relocate function provides a confusing, insufficient and unsafe API. It works with raw pointers (safety issue), it does not handle potentially throwing moves (which makes it insufficient for usage within general containers), and it steps on the toes of the uninitialized algorithms (P3516) that WG21 agreed was the correct user-facing API. P3516 was design-approved but missed C++26 due to limited LWG bandwidth in Sofia. We should remove std::relocate instead of introducing an API that we know to be problematic and already have a designed successor for.	Remove std::relocate from [obj.lifetime].	
RO 1-134	9-15	20.2.6		te	Although <code>trivially_relocate</code> was intended as the most fundamental operation for trivial relocatability, it's insufficient for key functions like <code>realloc</code> , which may either move an object's data or leave it in place. A more primitive operation, <code>start_lifetime_at</code> , addresses this and related use-cases.	<p>Add the following to 20.2.6 [obj.lifetime]:</p> <pre>template T* start_lifetime_at(uintptr_t origin, void* p) noexcept; Mandates: is_trivially_relocatable_v && ! is_const_v is true. Preconditions: – [p, (char*)p + sizeof(T)) denotes a region of allocated storage that is a subset of the region of storage reachable through [basic.compound] p and suitably aligned for the type T.</pre>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						<p>– The contents of <code>[p, (char*)p + sizeof(T))</code> is the value representation of an object <code>a</code> that was stored at <code>origin</code>.</p> <p><i>Effects:</i> Implicitly creates an object <code>b</code> within the denoted region of type <code>T</code> whose address is <code>p</code>, whose lifetime has begun, and whose object representation is the same as that of <code>a</code>.</p> <p><i>Returns:</i> A pointer to the <code>b</code> defined in the Effects paragraph.</p>	
US 74- 135	9-15	20.2.6		te	Although <code>trivially_relocate</code> was intended as the most fundamental operation for trivial relocatability, it's insufficient for key functions like <code>realloc</code> , which may either move an object's data or leave it in place. A more primitive operation, <code>start_lifetime_at</code> , addresses this and related use-cases.	<p>Add the following to 20.2.6 [obj.lifetime]:</p> <pre>template T* start_lifetime_at(uintptr_t origin, void* p) noexcept;</pre> <p><i>Mandates:</i> <code>is_trivially_relocatable_v</code> && <code>! is_const_v</code> is true.</p> <p><i>Preconditions:</i></p> <p>– <code>[p, (char*)p + sizeof(T))</code> denotes a region of allocated storage that is a subset of the region of storage reachable through <code>[basic.compound] p</code> and suitably aligned for the type <code>T</code>.</p> <p>– The contents of <code>[p, (char*)p + sizeof(T))</code> is the value representation of an object <code>a</code> that was stored at <code>origin</code>.</p> <p><i>Effects:</i> Implicitly creates an object <code>b</code> within the denoted region of type <code>T</code> whose address is <code>p</code>, whose lifetime has begun, and whose object representation is the same as that of <code>a</code>.</p> <p><i>Returns:</i> A pointer to the <code>b</code> defined in the Effects paragraph.</p>	
CA- 136		20.2.6	Paragraph 16	te	Const objects can be destroyed and const objects can also be created therefore requiring	Allow const-qualified types for <code>`relocate`</code> by striking the <code>`!is_const_v<T>`</code> mandate.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					`!is_const_v<T>` for `relocate` is overconstraining.		
CA-137		20.2.6	Paragraph 9	te	Const objects can be destroyed and const objects can also be created therefore requiring `!is_const_v<T>` for `trivially_relocate` is overconstraining.	Allow const-qualified types for `trivially_relocate` by striking the `!is_const_v<T>` mandate.	
US 75-138		20.3.2.2 20.3.2.3 20.3.2.4		te	There seems to be few if any valid constexpr use of owner_before outside of extremely contrived examples.	Reconsider whether it makes sense for owner_before and owner_less to be constexpr.	
US 76-139		20.3.2.2.7	13,19,28	te	These attempt to form the type U[N] which is not a valid type when N == 0.	Specify what happens in that case.	
US 77-140		20.4.1		te	indirect<T> should convert to T& to simplify the use cases (e.g., returning the object from a function with a return type T&) where indirect<T> appears as a drop-in replacement for T when T may be an incomplete type conditionally. With the proposed change, indirect<T> is closer to reference_wrapper<T>, but carries storage	Add (constexpr and noexcept) operator const T&() const &, operator T&() &, operator const T&&() const &&, and operator T&&() &&.	
GB06-141		21		te	Support emitting messages at compile-time. With the greatly increased support for compile-time programming (constexpr exceptions, constexpr containers, reflection, ...) it is more important than ever that we can debug compile-time code. There is currently no support for this.	Adopt P2758 “Emitting messages at compile time”	
FR-007-011-142		21.2		te	While we have expansion statements and the ability to introduce packs in C++26, we cannot use them with std::integer_sequence. That makes them significantly less useful. The remedy is to give the type support for the structured bindings protocol, as proposed by P1789R1, which was already forwarded by LEWG.	Adopt P1789R1.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
CZ2- 143		21.2 [intseq]	whole section	te	Index sequence should be destructurable as proposed in P1789R1, and this functionality is missing to improve metaprogramming facilities. And it was missing in proposal P1061R10 which added ability to introduce pack in structured binding.	Adopt changes proposed by P1789R1.	
US 78- 144		21.2.2		te	With the adoption of allowing packs in structured bindings and expansion statements, it now becomes very useful to have <code>std::integer_sequence</code> opt into structured bindings. Failing to do so makes it behave as a type with no members, which is a very surprising outcome.	Adopt P1789.  p1789r1.pdf	
US 82- 145		21.3.12		te	P2641 added the function <code>std::is_within_lifetime</code> , but it is too specific. A very slight generalization allows it to also check for derived classes being within their lifetime, which is significantly more useful.  p2641r4.html	Adopt P3450.  p3450r0.html	
US 79- 146		21.3.3		te	The “cw-fixed-value” construction in the default template argument for the second template parameter of <code>constant_wrapper</code> is redundant.	Delete it.	
RU- 147		21.3.3	Header <type_traits> synopsis [meta.type.s ynop]	te	Misuses if type traits cause UB and hard to debug errors. Put time into solving library issues 3099 and 3022 and consider hardening the compile time preconditions		
US 80- 148		21.3.5		te	The trailing requires-clause of <code>operator()</code> has an unused (Args...).	Remove it.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 81- 149		21.3.6.4	Table 54	te	The precondition for <code>is_consteval_only</code> is insufficient. A definitive false answer requires every type that T is compounded from to be complete or cv void.	Modify the precondition accordingly.	
US 85- 150		21.4		te	The reflection APIs are specified to throw an exception, but the contents of that exception are insufficiently precise. We should specify that the exception thrown has its <code>from()</code> populated from the API that failed.	Adopt P3795R0, section 2.7  p3795r0.html	
US 84- 151		21.4		ed	In some examples <code>static_asserts</code> are presented without comment while others they are presented with the comment <code>// OK</code> .	Make them consistent.	
US 83- 152		21.4		ed	In many cases a <i>Throws:</i> or a <i>Remarks:</i> paragraph is separated from the rest of the specification of the function by a long example, making it easy to miss.	Move the examples to the end.	
RU- 153		21.4	Reflection [meta.reflect ion]	te	Reflection allows stateful metaprogramming, however Core issue 2118 "Stateful metaprogramming via friend injection" still tries to prohibit it. Consider explicitly stating that “it is fine” as some of the popular applications rely on friend injection stateful metaprogramming	Consider closing Core issue 2118 "Stateful metaprogramming via friend injection" as NAD.	
BDS1 -154		21.4 [meta.refl ection]		te	Reflection functions in <code>[meta.reflection]</code> are specified to throw <code>std::meta::exception</code> , but the contents of the thrown exceptions aren't specified.	Add a paragraph to <code>[meta.syn]</code> with the following contents: When a function or function template F specified in this header throws a <code>meta::exception E</code> , <code>E.what()</code> is a string	

¹ MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						describing the error, E.from() is a reflection representing F, and E.where() is the source location of the call to F.	
FR-017 -155		21.04.01		Te	meta::has_ellipsis_parameter is a confusing name, as it could refer to either a C-variadic function, or to a variadic C++ template parameter.	Introduce a core term to describe functions with such an ellipsis - (maybe c-variadic function) and rename meta::info::has_ellipsis_parameter to be reflective of that term	
US 87-156		21.4.1 21.4.18		ed	The annotations functions should be grouped with the other queries instead of being placed at the end of 21.4.	Move 21.4.18 to after 21.4.11 and likewise update the synopsis.	
US 86-157		21.4.1 21.4.3 21.4.15		ed	The title of 21.4.3 is misleading since two of the three functions are not about to strings. Additionally, the specification in 21.4.3 is heavily dependent on the functions specified way below in 21.4.15.	Merge 21.4.3 and 21.4.15 into one subclause "Promoting to static storage". Update the synopsis in 21.4.1 accordingly	
PL-010		21.4.1	[meta.syn]	te	<p>The functions inside the std::meta should not be marked as addressable functions. This prevents us from changing their signature or providing additional overloads with the same name in the future, as this would break existing code that passes &std::meta::function to algorithms.</p> <p>For illustration, currently, traits like is_constructible_type do not allow you to define the context in which the constructibility should be checked and use the equivalent of std::access_context::unprivileged, i.e., private members are not usable.</p> <p>In the future standard, we could add additional overloads accepting access_context arguments that would allow us to inspect if the class has a private constructor.</p>	<p>Remove [meta.syn] p1:</p> <p>Unless otherwise specified, each function, and each specialization of any function template, specified in this header is a designated addressable function ([namespace.std]).</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FR-012 -158		21.4.1		te	that <code>std::meta::is_class_type</code> would error when the parameter is not a type is confusing, and would lead to needlessly verbose and cumbersome usage of reflection interfaces. This problem is fairly well described by P3781	Adopt something along the lines of what is proposed by P3781	
FR-013 -159		21.4.1		ge	It is not clear how range producing methods such as <code>members_of</code> are supposed to be implemented efficiently. A compiler cannot manipulate <code>std::vector</code> directly, as they are fairly complex types with large implementation divergence. And we want to avoid going from the constant evaluation domain to compiler internals for each reflected element as this would be inefficient. Instead, to implement <code>members_of</code> an implementation should either - Construct a vector of the correct size in an uninitialized state and pass <code>data()</code> to the compiler (which would require 2 round trips between constant evaluation and compiler internals) - Have a vector interface that takes ownership of a pointer + size Given the number of compilers and standard libraries that need to interoperate, it would have been useful to standardize a way to construct a vector from a range of reflection efficiently.	Adding a constructor to <code>std::vector</code> that would take ownership of an allocation would facilitate implementation of reflection. Such a constructor could be exposition-only but it should be consistent across implementations. We would prefer reflection to keep using <code>std::vector</code> rather than a new type such as proposed by P3429	
FR-014 -160		21.4.1		te	<code>define_static_string</code> and <code>define_static_object</code> exist as a library solution to limitations to the set of types that are allowed to exist as template parameters. However, while these functions technically solve the problem, it's unlikely that they would be useful	- make <code>string_view</code> structural - make span of structural types structural (a compiler can ignore access for these types) - consider whether we could make <code>std::string</code> and <code>std::vector</code> of structural types structural in C++26. The general problem of making	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					long term and they add undue complexities to common reflection use cases.	vectors of arbitrary types, or to make arbitrary containers structural does not have to be solved to make reflection more useful.	
FR-015 -161		21.4.1			It is not clear how meta::has_default_argument should interact with CWG2701	remove meta::info::has_default_argument from C++26 and consider a holistic solution to the problem of reflecting on composants that can change between different declarations of the same entity.	
FR-016 -162		21.4.1			<p>C++26 allows reflecting on function parameter names.</p> <p>Parameter names are historically not part of the interface of libraries, as they were only referenced inside functions.</p> <p>Allowing reflection on function parameter names extends the set of properties observable in existing code that was never designed with that possibility.</p> <p>Given Hyrum's law, this aspect of reflection in particular (and the extension of what are the constituent parts of a library interface in general) will cause undue burden on library writers.</p> <p>We would prefer solutions where</p> <ul style="list-style-type: none"> - parameters are manipulated through their indexes - there is some opt-in mechanism for libraries that want to render their names visible. 	Disallow reflecting on parameter names in C++26	
US 88-163		21.4.1	2	ed	In Note 2, "The behavior ...have semantics" is not grammatically correct.	Strike "The behavior of".	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 103- 164		21.4.10	6	ed	There is an extraneous “of” before “B”.	Strike it.	
US 104- 165		21.4.11	2	ed	“direct base class relationship” is not an entity.	Change to “construct” or similar.	
US 105- 166		21.4.11	5	ed	Bullet 5.3 does not work with the introductory text of the list.	Move it out of the list.	
US 106- 167		21.4.11	6	te	“W is not \perp ” should be “W is \perp ” as the goal is to exclude bit-fields.	Strike “not”.	
US 107- 168		21.4.11 21.4.18	6, 8, 10 5	ed	21.4.11 bullets 6.2, 8.2, and 10.2 and 21.4.18 bullet 5.2 are saying the same thing but in two different ways.	Change 21.4.11 bullet 10.2 and 21.4.18 bullet 5.2 to use the same words as 21.4.11 bullet 6.2 and 8.2.	
US 108- 169		21.4.11	7	ed	In bullet 7.5, A is already a value; it does not make sense to ask for “the value of A”.	Strike “of”.	
US 109- 170		21.4.11	8	te	This should similarly disallow data member descriptions of bit-fields.	Add “(T, N, A, W, NUA) (11.4.1) where W is \perp ” after “data member description”.	
US 110- 171		21.4.12	07.1	ed	There is an extraneous “T is” before “is_convertible_v”. Additionally, X is formatted differently (in code font in two places and not when it first appeared).	Strike “T is”. Format X consistently.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 112- 172		21.4.12	10	te	Bullet 10.3 should disallow derived-to-base conversions, just like bullet 10.1.	Add a requirement that remove_extent_v<U>* and T are similar types.	
US 113- 173		21.4.12	12	te	The second if branch should not be constexpr.	Strike "constexpr".	
US 111- 174		21.4.12	8, 10	ed	U is in code font in paragraph 10 but not in paragraph 8	Make them consistent.	
US 114- 175		21.4.13		te	This wording needs to be updated to account for the splice-specifier changes.	Resolve LWG issue 4316.	
US 115- 176		21.4.13	1, 5	ed	The wording should specify that the order of elements of Args is that of the elements in arguments. (This was in the incoming paper.)	Add ", in order" after "arguments" in paragraphs 1 and 5.	
US 116- 177		21.4.13	9	ed	The first error message could be clarified to explicitly say that the undeduced placeholder type is the return type.	Reword as: // error: fn<int> contains an undeduced placeholder type for its return type.	
US 117- 178		21.4.14	4	ed	Does the class TCIs need to be marked as exposition-only?	Move the intended class template TCIs to the end of p2: "Let TCIs be the invented template template<T P> struct TCIs;, strike ", with TCIs as defined below" from p3, and "given the invented template template<T P> struct TCIs;" from p4.	
US 118- 179		21.4.14	8, 11	te	This should talk about the object/function designated by expr/fn, rather than expr/fn.	Reword accordingly.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 119- 180		21.4.15	1	ed	The name and (1) seem incorrect here. The section is called "Promoting to static storage arrays", similar to 21.4.3, but it's doing strings as well. (1) is also a copy-paste of the overview of 21.4.3 (1). This section and 21.4.3 should also be merged or put alongside each other.	Merge this clause with the (renamed) 21.4.3 above.	
US 120- 181		21.4.15	10	te	It is not clear what <code>ei</code> is when proxy references are involved.	Specify that it is <code>T</code> .	
US 121- 182		21.4.15	11	te	The initialization of <code>P</code> uses copy-initialization but the Mandates clause uses direct-initialization.	Tighten the requirements.	
AT5- 183		21.4.15	Paragraph 11	te	The return type of <code>reflect_constant_array</code> is currently inconsistent as <code>std::array</code> is used for zero-length results and language-level arrays are used for all other sizes.	Rewrite paragraph 11 as: Let P be the template parameter object (13.2) of type <code>const array<T, sizeof...(V)></code> initialized with <code>{[:V:]...}</code> .	
US 122- 184		21.4.16		te	<code>data_member_spec()</code> separates the type from every other option. But the name is almost always mandatory, which makes the usage strange. The type should be moved into the rest of the options.	Adopt P3795R0, section 2.3.	
FR- 018- 185		21.4.16		Te	<code>define_aggregate</code> has a very narrow set of use cases, and a very novel interface that offers limited options and use cases. That interface, whose usage is very different from the rest of C++ will likely be obsoleted in the near future, and will just become a maintenance burden. That interface is also not very extensible and it seems it is solely motivated by the desire to ship a barely usable form of generative meta programming in 26. It's also one of the most problematic aspect of reflection/P2996 as there remain questions around	Remove <code>define_aggregate</code> (and <code>data_member_options</code> , <code>data_member_spec</code>) form C++26	

1 MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



2 Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					complete class contexts, reachability and lookup (even if consteval blocks solved some of these issues)		
US 124- 186		21.4.16	05.1	te	This function throws if "dealias(type)" is either and object type or reference type. However, the worded appears to just assume that the given "meta::info type" value represents a type at all.	Specify that is_type(type) must be true as a precondition, or modify the wording to throw an exception if this is false.	
US 123- 187		21.4.16	2, 3	ed	The declaration is neither meaningful C++ nor fragment from the class definition.	Either qualify as name-type::name-type or do not qualify at all if it is intended as a "quote" from the class definition.	
US 125- 188		21.4.16	7	ed	Similar to the above comment; is the wording here correct to ensure that C in indeed a reflection of an incomplete class type? Should it be written as a "Mandates:" requirement, making the program ill-formed?	Add a new paragraph after [meta.reflection.define.aggregate]p7: Mandates: C shall be an incomplete class type.	
US 126- 189		21.4.16	8	te	This should clarify that, despite the completeness of C affecting the semantics of the program, no implicit instantiation is performed not withstanding 13.9.2.	Add an exception to the general rule, possibly as a new <i>Remarks</i> paragraph.	
US 127- 190		21.4.16	8, 9	te	N _k is defined as an identifier (see 11.4.1) and should not be compared with code or with string literals in bullet 8.4. Similarly, 9.5.1 should not talk about "character sequence encoded by N _k "	Reword the specification to be consistent with the language definition.	
US 129- 191		21.4.17		te	P2996 added a large number of consteval functions named is_meow_type. Those currently throw an exception when passed a reflection not representing a type, but that is pretty surprising and user-hostile. They should simply return false for non-types, as the names suggest.  p2996r13.html	Adopt P3781.  p3781r0.html	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 128- 192		21.4.17		te	P2996 added consteval metafunctions for every type trait in the standard library, except for three new ones that were adopted in Sofia: <code>is_applicable</code> , <code>is_nothrow_applicable</code> , and <code>apply_result</code> . Those should be added as well.	Adopt P3795R0, section 2.5  p3795r0.html	
US 130- 193		21.4.17	Table 64	te	This table does not say what happens if <code>std::FOO_t<...></code> does not exist.	Specify that an exception is thrown.	
DE- 194		21.4.17 [meta.reflection.traits]		te	Reflection traits with a name of form " <code>is_*_type</code> " should imply trait ' <code>std::meta::is_type(std::meta::info)</code> ' as the name of the trait implies. Such kind of a trait can be answered with a clear 'yes' or 'no' - leaving no room for anything else, such as raising an exception. Paper P3781r0 expands on that and gives details, including a list of traits which are affected, and which are not.	Do not apply clause 2 to said traits (as listed in P3781r0), making them free of preconditions.	
US 131- 195		21.4.18		te	Annotations are not allowed on function parameters, due to annotations and function parameter reflection being added in parallel. They should be allowed.	Adopt P3795R0, section 2.4  p3795r0.html	
US 89- 196		21.4.3		ed	The name of the section is "Promoting to static storage strings". This makes sense if the only function was <code>define_static_string</code> , but there is also <code>define_static_array</code> and <code>define_static_object</code> . The segment name should be more general.	Rename the segment to "Promoting to static storage".	
US 90- 197		21.4.3	4-5	te	template consteval const remove_cvref_t* <code>define_static_object(T&& t);</code> should be revised to return <code>const T&</code> instead <code>const T*</code> since it is never <code>nullptr</code> . Current pointer interface requires usage to constantly dereference with <code>*</code> even the default usage in input parameters is <code>const T&</code> . automatic and static storage duration is by default not <code>nullptr</code> . A pointer interface implies the possibility	https://isocpp.org/files/papers/P3739R2.html  p3739r2.html	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					of nullptr which doesn't seem to be the case here. If nullptr was a possibility, which it isn't, then const optional<&> should have been used instead.		
US 91- 198		21.4.5	2	ed	There is an extraneous "the" before "operators".	Strike it.	
US 92- 199		21.4.5	Table 63	ed	The header row should have borders on the left and right.	Add them.	
US 93- 200		21.4.6		te	source_location_of seems inconsistent with other query functions that may not have a sensible return value. Some of these other functions have a corresponding function for querying if the call would be valid. For example, identifier_of has the corresponding has_identifier function, and template_arguments_of has has_template_arguments.	Add a has_source_location function. source_location_of can continue to return a default value, or throw an exception like identifier_of.	
US 94- 201		21.4.6	3	ed	The font of <i>N</i> in bullet 3.6 should be consistent between the two occurrences.	Make them consistent.	
US 95- 202		21.4.7		te	There are a few simple predicates that were overlooked in the adoption of P2996, that are consistent with other simple predicates that were present: is_inline, is_constexpr, and is_consteval.	Adopt P3795R0, section 2.1.  p3795r0.html	
US 97- 203		21.4.7	25	te	Language linkage is a property of functions, variables, and function types (9.12), not of names.	Respecify has_c_language_linkage according to the language definition.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 98- 204		21.4.7	26	te	Years of experience with traditional type traits have repeatedly demonstrated that <code>is_complete_type</code> is an extremely dangerous meta function that naturally leads to ODR violations and other IFNDR conditions. There is a strong demand for an <code>assert_is_complete</code> facility that should be added to replace this specific function.	Remove the <code>is_complete_type</code> meta function and add a new meta function, <code>assert_is_complete_type</code> that throws a <code>meta::exception</code> unless passed a reflection to a complete type. Revise the three applications of the <code>is_complete_type</code> meta function in the meta functions library to fail in the same way, ideally by using the new <code>assert_is_complete_type</code> meta function.	
US 99- 205		21.4.7 21.4.11	50	te	The <i>Throws</i> paragraph on dealias is inconsistent with its use in 21.4.11 (e.g., 21.4.11 paragraph 6 contemplates that dealias(r) can represent various things that are not entities).	Either remove the restriction and specify that in his case the input is returned unchanged, or edit 21.4.11 to not use dealias.	
US 96- 206		21.4.7	7	ed	Second comment in example 2 should say <code>constant_of(^x)</code> instead of <code>constant_of(x)</code>	Add the missing <code>^^</code> .	
US 100- 207		21.4.8		te	<code>std::meta::access_context</code> provides a mechanism for getting the current scope, which is a useful and occasionally asked for piece of functionality. But it provides it in a very surprising and indirect way. We should provide this functionality in a more discoverable, direct way.	Adopt P3795R0, section 2.2.  p3795r0.html	
US 101- 208		21.4.8	11	ed	The declaration of <code>via</code> has an extraneous <code>static</code> .	Strike it.	
US 102- 209		21.4.9 21.4.18	6, 8 5	te	“is a constant (sub)expression” is incorrect now that errors are reported via exceptions	Change to “the evaluation of...would not exit via an exception”.	
FR- 019 -210		22.2.1		te	The design space of constant evaluated function parameters, <code>constant_wrapper</code> , <code>nontype_t</code> and its integration in <code>function_ref</code> has not been fully explored.	Remove <code>constant_wrapper</code> , <code>cw</code> and <code>nontype_t</code> from C++26. (<code>function_ref</code> 's abi can be made forward compatible by implementers)	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					We are concerned that the many proposals trying to improve the ergonomics of these types have not been fully explored. In particular it is concerning that <code>constant_wrapper</code> is not suitable for the needs of <code>function_ref</code> . It would be prudent to delay this work to a future version of C++.	Explore language-level solutions (<code>constexpr</code> parameters) in future versions of the standard.	
US 132- 211		22.3.2	20, 23, 32, 36	te	Unlike the more sophisticated optional and variant, <code>pair</code> never has trivial assignment operators. This surprise interferes with metaprogramming and with trivial copyability (under the current, restrictive definition).	Specify that the copy and move assignment operators are trivial when appropriate. Alternatively, broaden the core-language definition of trivial copyability to reduce the impact of the library design.	
US 133- 212		22.4.4.3	2, 5, 8, 12	te	As with <code>pair</code> , <code>tuple<int></code> and even <code>tuple<></code> are not trivially assignable and thus not trivially copyable.	As for <code>pair</code> .	
AT7- 213		22.4.7	Paragraphs 6 and 8	te	<code>complex</code> has been made <i>tuple-like</i> (see 22.4.3 and 29.4.9), but the current wording does not support <i>cv-qualified</i> <code>complex</code> .	Add <code><complex></code> to the list of headers that make the <code>const</code> -specialization available.	
PL- 011		22.5	[optional]	te	The range support was added to the optional, making it usable with range adaptors defined in <code>std::views</code> , however, we have not updated the views specification to handle it optimally when possible. This leads to unnecessary template instantiations.	Add a special case to recognize <code>optional<T></code> for adaptors: <ul style="list-style-type: none"> views::as_const: should return <code>optional<const T></code> or <code>optional<const U&></code> (if <code>T</code> is <code>U&</code>) views::take(opt, n): empty <code>optional<T></code> if <code>n</code> is equal to zero, opt otherwise views::drop(opt, n): empty <code>optional<T></code> if <code>n</code> greater than zero, opt otherwise views::reverse: input unchanged	
FR- 020 -214		22.5.2		te	As described by P3415, treating a kind of object (such as optional) as a different kind of entity a range can lead to breakage and incongruent	Remove P3168R2 from C++26. Continue work on a <code>maybe_view</code> , or other explicit coercion to ranges such as rust's <code>as_range</code>	

1 MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>behavior in generic code - notably in any printing, serialization etc code.</p> <p>For example, there is no justification given as to why a non-engaged optional should be formatted as a range.</p> <p>The standard side step that question by disabling that formatting in [optional.syn]. However this is a luxury that is not afforded to other libraries and will make adoption of C++26 harder.</p> <p>For a type to behave as a range we should be willing for it to behave as a range by default by default in all contexts.</p> <p>Because an optional is more closely similar to a pointer, than a range, should all pointers and pointer-like-objects behave like ranges in all contexts?</p> <p>The notion of platonic types, described in P0705 is fundamental to a healthy ecosystem of generic components. If any type can behave as something that it isn't, composition becomes surprising, or impossible.</p> <p>"f we are to succeed in producing widely reusable components, idiosyncratic interfaces are no longer usable. A component programmer must be able to make some fundamental assumptions about the interfaces she uses, without ever seeing their implementations or even imagining their applications. " - Stepanov.</p>		
US 134- 215		22.5.4.1		te	optional<T&>, has a typical implementation which just wraps a pointer, and so should be trivially copyable.	Add the sentence "optional<T&> is a trivially copyable type."	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 135- 216		22.8.6.4	2, 5	te	As with pair, no specialization of expected is trivially assignable and thus not trivially copyable.	As for pair.	
US 136- 217		22.8.7.4	1, 4	te	As with the primary template, no expected<void,E> is trivially assignable and thus not trivially copyable.	As for the primary template.	
FR- 021 -218		22.10		te	Don't rename std::nontype_t to std::constant_arg_t as proposed by P3774R0. Instead, rename it to something like std::function_wrapper and make it callable, as originally proposed by P3774R1, by adopting P3843R0.	In decreasing order of preference: <ol style="list-style-type: none"> 1. Adopt P3843R0. 2. Keep the status quo in the working draft (std::nontype_t); do not adopt P3774R1. 3. Remove std::nontype_t and the std::function_ref constructors using it. 4. Any other solution in the problem space. Adopt P3774R1.	
US 137- 219		22.10.6.1	2	te	The statement that reference_wrapper is trivially copyable seems to contradict the synopsis.	Indicate that the relevant special member functions are defaulted, perhaps by introducing the obvious exposition-only non-static data member.	
RU- 220		22.10.17.01	General [func.wrap.g eneral]	te	Allow skipping indirection in function_ref for a more efficient implementation	Consider applying the fix from library issue 4264	
RU- 221		22.10.17.05	Copyable wrapper [func.wrap.c opy]	te	`copyable_function()` and `copyable_function(nullptr_t)` can construct the object without any dynamic initialization, leading to better runtime efficiency and avoiding	Consider adding `constexpr` to `copyable_function()` and `copyable_function(nullptr_t)` in [func.wrap.copy.class] and in [func.wrap.copy.ctor]	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					all the potential initialization order issues.		
RU- 222		22.10.17.06	Non-owning wrapper [func.wrap.ref]	te	`function_ref(F*)` can construct the object without any dynamic initialization, leading to better runtime efficiency and avoiding all the potential initialization order issues.	Consider adding `constexpr` to `function_ref(F*)` constructor in [func.wrap.ref.class] and in [func.wrap.ref.ctor]	
PL- 005		22.10.17.6.5	[func.wrap.ref.deduct]	te	As function_ref supports noexcept specification, it should be properly deduced by class argument deduction. Currently, this is not true for data member pointers, as for: M C::* dm; C c; The function_ref(dm, c) deduces function_ref<M&()>, instead of function_ref<M&() noexcept>.	Modify [func.wrap.ref.deduct] as follows: (6.2) F is of the form M G::* for a type G and an object type M, in which case let R be invoke_result_t<F, T&>, A... be an empty pack, and E be >false<ins>true</ins>, or	
GB07 -223		22.12		te	Add feature test macro for stdbit.h The __STDC_VERSION_STDBIT_H__ macro defined by C is not sufficient to check if a C++26 implementation provides a usable <stdbit.h> header. If a libc header of that name is found somewhere in the C++ compiler's include paths (e.g. in /usr/include) then it might be included and would define the macro. However, the libc version of the header might use C-specific features such as _Bool or _Generic and cause errors in a C++ program. A C++-specific macro would only be present in a C++-aware header, and so a C++ program could check that to be sure that the header is usable in C++ code.	Add a new <code>__cpp_lib_stdbit_h</code> macro	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 138- 224		23.2.2.5, 23.3.15, 23.3.16		te	<p>The new container <code>std::inplace_vector<T,Cap></code> does not take an allocator parameter. This makes it inconsistent with every other STL container that manages object lifetimes. (The two C++23 "containers" that lack the allocator parameter are <code>std::array</code>, which does not need to construct or destroy objects during its lifetime; and <code>std::tuple</code>, likewise. <code>std::tuple</code> nevertheless has had to develop an entirely parallel-universe constructor API to deal with allocators. This indicates that allocators can't be avoided in the modern STL.)</p> <p>P3160 shows several examples where <code>inplace_vector<T,Cap,A></code> (such as <code>pmr::inplace_vector<T,Cap></code>) is required. For example, you might like to replace one use of <code>std::pmr::vector<std::pmr::string></code> with <code>std::pmr::inplace_vector<std::pmr::string, 10></code> for performance; but without an allocator, you can't <code>push_back</code> new elements into that <code>inplace_vector</code>; the <code>inplace_vector</code> user must do <code>v.emplace_back("abc", &mr)</code> and pass around the allocator parameter separately from the container that uses it, but vice versa that strategy does NOT work for the <code>pmr::vector</code> user because he is not allowed to do <code>v.emplace_back("abc", &mr)</code></p> <p>This makes <code>inplace_vector</code> not a drop-in replacement for <code>std::vector</code>; it is not composable in generic code with the rest of the STL's containers.</p> <p>P3160 shows that adding an allocator parameter to <code>inplace_vector</code> can be done with zero runtime cost, zero memory cost, and effectively zero compile-time cost (i.e., we follow the existing practice for existing STL containers, and suffer no penalty that the existing containers don't already suffer). There is a reference</p>	<p>Adopt P3160R2 "Allocator-aware <code>inplace_vector</code>".</p>  <p>p3160r2.html</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					implementation of allocator-aware <code>inplace_vector</code> in the SG14 repository.		
GB08 -225		23.3.16		te	<p><code>inplace_vector::try_xxx</code> functions should return <code>optional<reference></code></p> <p>The <code>try_emplace_back</code> and <code>try_push_back</code> members of <code>inplace_vector</code> should return <code>optional<reference></code> instead of pointer. Those functions are closely related to <code>push_back</code> and <code>unchecked_push_back</code> which return references, so the <code>try_push_back</code> form which can fail should return an optional-reference, i.e. <code>optional<reference></code>. A pointer is not just a nullable reference.</p> <p>Returning <code>optional</code> also allows the monadic operations (<code>and_then</code> etc) to be used to take actions based on whether the push back was successful.</p>	Change the return type of <code>try_emplace_back</code> and <code>try_push_back</code> from <code>pointer</code> to <code>optional<reference></code> . Change the Returns: element to say: <code>nullopt</code> if <code>size()</code> is equal to <code>capacity()</code> , otherwise <code>back()</code> .	
US 149- 226		23.3.16		ge	There are concerns over redesigning <code>inplace_vector</code> .	See https://wg21.link/P3830  p3830r0.pdf	
FR- 022 - 227		23.3.16		te	<p><code>inplace_vector</code> is specified to throw <code>bad_alloc</code> when size exceeds capacity. However, <code>inplace_vector</code> does not allocate, and exceeding the capacity does not constitute an allocation failure.</p> <p>This is important because <code>bad_alloc</code> is often caught for either system wide monitoring of a system resource (memory)- or to set up some sort of "liberate memory" and retry schemes in some allocators. None of these solutions would be applicable to <code>inplace_vector</code>.</p> <p>While we would prefer <code>inplace_vector</code> interfaces to have preconditions by default (not just in the <code>unchecked_</code> functions), using any exceptions other than <code>bad_alloc</code> would be an improvement</p>	<p>modify <code>inplace_vector</code> to throw a different exception when size exceeds capacity.</p> <p><code>std::length_error</code> or a new exception type derived from either <code>std::logic_error</code> or <code>std::runtime_error</code> would be appropriate.</p> <p>If, as claimed by P3830, issues with <code>inplace_vector</code> should not be addressed in C++26, we should delay the standardisation of <code>inplace_vector</code> to a future version of the standard.</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 150- 228		23.3.16.5	8-14	te	inplace_vector's try_emplace_back and 2 try_push_back should be revised to use const optional<&> instead of pointer as it is safer from null pointer dereference, pointer arithmetic and indexing issues	https://isocpp.org/files/papers/P3739R2.html  p3739r2.html	
AT6- 229		23.3.3.5	Paragraph 3	te	This paragraph makes calling front/back for zero-length arrays UB. New safety features ("standard library hardening") would make this a hardened precondition (see 23.2.4.72 and 23.2.4.76). Hardened preconditions are preferable to UB.	Strike this paragraph (effectively applying the resolution to LWG4276).	
NC IT- 230	-	23.3.8	--	Te	P0447 added std::hive which is a very narrow-scope, performance-oriented, library-only container. An ABI stable implementation will not be able to apply all the performance optimizations needed to maintain the efficiency of this container in future versions, while an external library will not need to guarantee ABI stability, giving the implementors more freedom to optimize their code. A standard implementation won't be competitive with an external implementation, so the usefulness of a standard implementation would be limited.	Revert P0477	
CZ1- 231		23.3.8 [hive] to 23.3.9.6	multiple places of [hive]	te	"hive" container is not marked constexpr, as it was merged after P3372R3 which made all other containers marked with the "constexpr" qualifier. It makes the section not symmetric in intent.	Mark all functions and member functions with constexpr qualifier for symmetry of with rest of 23 [containers] as there is no technical reason for this container be only not marked constexpr.	
US 139- 232		23.3.9.1	05.4	te	Paragraph 5.4 allows for UB when the user doesn't manage to keep the block limits within bounds and the min below the max, yet no tools are offered to check these conditions easily. In terms of providing safe facilities this UB is an easy trap to fall into. The methods that allow to set block limits on hive should rather check the	Adjust all constructors and methods on hive to throw in case specified block limits violate the conditions set out here. Or at least provide a function to the user to correctly check the condition separately.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					conditions and throw in case of violation or at least some function should be provided to let the user check the conditions easily.		
US 140- 233		23.3.9.2		te	For the constructors and assignment operators that can construct a hive with a sequence of elements, only one uses wording that prescribes the resulting order of elements in the new object. For a pair of iterators first/last, the effects say the hive will be equal to the range. For other constructors it only says that the elements are the same without explicitly prescribing the order. Given that the introductory matter for hive mentions that the container chooses insertion points for elements, it's unclear whether that would mean the order is unspecified for such constructors. Anyway it's unclear why the wording is different.	Insert wording to say "Constructs a hive object with the elements of x with the same order as they appear in x" and possibly adjust the wording for first/last in that fashion too. Applies to paragraphs 11, 13, 16, 19, 23, and assignment operators in 26 and 29. Or otherwise remove the ordering guarantee from the iterator-based version in order to match the other constructors. Then it's maybe worth to say the elements of x are inserted in an unspecified order.	
US 145- 234		23.3.9.3	12	te	The function trim_capacity(size_type n) has the effect of reducing capacity to no less than n. It doesn't explain the behavior in case where the capacity is less than n before the call. The condition isn't stated as a Postcondition, but still doesn't clear the confusion.	Change the Effects to include "if capacity() > n, capacity() is reduced to no less than n, otherwise no effect".	
US 141- 235		23.3.9.3	3	te	The description for reserve(n) mentions that the size of the sequence isn't changed, that iterators aren't invalidated, but doesn't say that the elements keep their order. Conceivably the implementation could rearrange blocks and possibly change the order of elements. Both reserve and trim_capacity could state in their remarks that the sequence is unchanged (which implies the size doesn't change).	Add to the Remarks of reserve(n) and trim_capacity that the sequence of elements doesn't change.	
US 142- 236		23.3.9.3	4	te	For "reserve(size_type n)" the postcondition states that "capacity() >= n" and the throws-clause applies only for "n > max_size()". This does not account for cases where the current "max_size() - capacity()" cannot be covered with	Change the postcondition to "capacity() >= n is true or capacity() >= max_size() - block_capacity_limits().min is true".	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					block sizes within the configured limits and n is less than a block away from max_size. For example when max_size=100, capacity=80, block min/max both are 40, and n=90. Furthermore, for cases where there would be a solution it could restrain implementations too much to be required to find it. When "max_size - n" is less than the minimum block size the implementation should either be allowed to throw or be allowed to deliver capacity less than the requested n, but at most one block away.		
US 144- 237		23.3.9.3	9	te	The functions that allow for "reallocation" state that in such case all iterators and references will be invalidated. The user, however, has no way to observe whether or not reallocation took place, other than to possibly create a special allocator type to observe such reallocation. So either the user has to assume that all iterators are invalidated on every call or they are given an indication for example through returning a bool.	Change the return type for shrink_to_fit and reshape to bool and add a Returns clause to say "true if reallocation took place, false otherwise".	
US 143- 238		23.3.9.3	9	te	The descriptions for shrink_to_fit and reshape state that in case of exceptions other than for block allocation, the effects are unspecified. However, these operations don't have reason to assign or swap any elements, thus leaving only copy or move construction for the "reallocation". When these exit with an exception (from the user supplied type) it's in the user's hand to care for exception safety on their types. So the contract could state that reallocation only will happen through those constructors and the destructor, enabling the user to ensure no elements get lost, only their order being unspecified in case of those exceptions.	Add to the Effects that reallocation happens only through constructing and destroying elements such that exiting such a constructor by exception will propagate the exception, leave the size of the sequence as before, and leave the elements in an unspecified order.	
US 146- 239		23.3.9.4	8	te	The function insert_range(rg) has effects to insert copies of the elements in rg. If the range iterators return elements as rvalue-references this should likely hit move-construction. Otherwise we	Change the Effects to say "Inserts the elements of rg." This would also be subject to the comment about the ordering guarantee.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					couldn't use it on move-only types such as <code>unique_ptr</code> . The corresponding wording for constructors avoids the word "copy" and says the elements are inserted.		
US 147- 240		23.3.9.5	3	te	The function splice could be more specific if in case of a <code>length_error</code> the containers are left as is or could possibly have changed. That is basically to say whether the block condition is checked upfront or while splicing blocks.	Add to the Effects "In case of an exception it is unspecified which elements of x are still in x or are now in *this, but all pointers and references stay valid."	
US 148- 241		23.3.9.5	8	te	The function template unique could state that in case of an exception the container has only removed an unspecified number of those elements that the predicate indicated to remove.	Amend the Effects with: "In case an exception is thrown it is unspecified which of the elements that the predicate so far indicated to remove are removed. Other elements are not removed."	
PL- 006		23.3.16.1	[inplace.vector.overview]	te	<p>As we now have optional support for references, the <code>try_push_back</code> and <code>try_emplace_back</code> functions of <code>inplace_vector</code> should return an <code>optional<reference></code>.</p> <p>This makes the API more consistent with the other overloads that return a reference. And allows the monadic interface to be used.</p> <p>Similarly the <code>try_append_range</code> should return an <code>subrange<iterator_t<R>, sentinel_t<R>></code>. This provides a more convenient interface, where a simple call to <code>empty</code> can be used to determine if all elements were inserted.</p> <p>Current:</p> <pre>auto range = createElemViews(); auto it = iv.try_append_range(range); if (it != range.end()) std::cout << "Elements dropped" << std::endl;</pre> <p>Proposed:</p> <pre>if (!iv.try_apend_range(range).empty()) std::cout << "Elements dropped" << std::endl;</pre>	<p>Change the return type of <code>inplace_vector</code> <code>try_push_back</code> and <code>try_emplace_back</code> functions to <code>optional<reference></code>.</p> <p>Change the return type to <code>try_append_range</code> function to:</p> <pre>conditional_t<ranges::borrowed_range<SR>, SR, ranges::dangling>;</pre> <p>Where SR is:</p> <pre>ranges::subrange<ranges::iterator_t<Rg>, ranges::sentinel_t<Rg>></pre>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 151- 242		23.7.2.1 29.10.2		te	We have two exposition-only concepts for similar things (integral-constant-like and constant-wrapper-like). The former can be express in terms of the latter.	Move constant-wrapper-like to the library introduction and update integral-constant-like to use it.	
US 152- 243		23.7.3.2, 23.7.3.7		te	<p>submdspan_mapping is a customization point, but submdspan_extents was and is not intended for users to customize. Adoption of P3663 for C++26 would add submdspan_canonicalize_slices, which again was and is not intended for users to customize. That makes three functions whose names start with "submdspan_", but only one of them is a customization point. It would make sense to distinguish the names that are not customization points. We propose subextents instead of submdspan_extents, and canonical_slices instead of submdspan_canonicalize_slices. We prefer "canonical" to "canonicalize" because the latter suggests an action applied in place to the arguments, rather than a function that takes some arguments and returns distinct objects of possibly different types.</p>  <p>p3663r2.html</p>	<p>After applying the changes in the latest version of P3663 (at least R3, which will follow LWG review currently in progress as of 2025/09/02), change all occurrences of submdspan_extents to subextents, and change all occurrences of submdspan_canonicalize_slices to canonical_slices. (This affects [mdspan.syn] and [mdspan.sub].)</p>  <p>p3663r2.html</p>	
PL- 007		23.7.3.7	[mdspan.sub]	te	<p>As currently specified, the strided_slice type has the following meaning:</p> <ul style="list-style-type: none"> * elements from range [s.offset, s.offset + s.extent) are referred, * s.stride defines the step in which the element is accessed. <p>As an illustration, given a 1-dimensional mapping sm produced by slicing mapping m with s:</p> <p>sm(i) is equal to m(s.offset + i * s.step)</p>	<p>Define the extent member of the strided_slice to represent the extent of the final slice object.</p> <p>(optionally) Provide an equivalent of the current functionality in the form of the additional slice_type, which is canonicalized to strided_slice.</p> <p>template<typename Offset, typename Extent,</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>As a consequence of the above, the extent (number of elements) of sm is equal to: $1 + (s.extent - 1) / s.stride$</p> <p>Due to the above, it is currently not possible to represent:</p> <p>1. stride value of 0, as computing the number of elements in sm, would require dividing by zero. A special case is currently where for s.extent equal 0, s.stride is ignored. This prevents us from defining broadcasting layouts.</p> <p>2. static extent of final mdspan in combination with dynamic stride value, as currently computing the number of elements in produced mdspan, requires knowing values of both s.extent and s.stride at compile time. Using a static extent value reduces the size of the extent object, and thus in consequence, mapping and mdspan.</p> <p>We should redefine the meaning of the extent member of the strided_slice to define the number of elements to a given extent in the produced mdspan. I.e. in the example defined above, the extent of sm would be s.extent, and elements from range [s.offset, s.offset + s.extent*s.stride) are referred.</p> <p>Such an approach naturally supports a zero s.strides value.</p> <p>Such a change should be done before C++26 finalization, as it would require the introduction of another canonical slice_type.</p>	<pre> typename Stride = constant_wrapper<1zu>> struct slice_range { Offset offset; Extent extent; Stride stride = Stride(); }; </pre>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
PL- 008		23.7.3.7	[mdspan.su b]	te	<p>The submdspan_extents name is excessively long and ties the functionality to creating submdspan. However, creating a subset of extents objects is generally useful in other contexts.</p> <p>The similarity of this name to submdspan_mapping also suggests that it may be a customization point that is invoked by ADL. However, the submdspan_extents is not specified to be one, as extents are not customizable.</p>	Rename submdspan_extents to subextents.	
PL- 009		23.7.3.7	[mdspan.su b]	te	<p>As currently specified, adding support for additional slice types to the submdspan function would break user-defined mappings and require changes in each mapping.</p> <p>We should decouple the user-facing interface of submdspan and layout implementers facing submdspan_mapping, by introducing a canonicalization step that will reduce the number of types that need to be handled later.</p>	Accept P3663R2.	
FR- 023 -244		25		te	Remove the reserve_hint members of view classes when their size members are already provided. Users are expected to call std::ranges::reserve_hint which doesn't need the member function when the range is sized.	Adopt P3763R0.	
FR- 024 -245		25		te	<p>During the evolution of P2846 – Eagerly reserving memory for not-quite-sized lazy ranges, the customization point function and member function names considered where size_hint, approximate_size, and reserve_hint, with the latter being adopted.</p> <p>However, the concept is still called approximately_sized_range, which is a mismatch. There is no indication that in order to become an approximately_sized_range, you have to add a reserve_hint member function, which is</p>	Rename reserve_hint to approximate_size.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>inconsistent with how becoming a sized_range requires a size member function. The name of the concept should reflect the name of the customization point.</p> <p>Because there does not appear to be a non-clunky concept name involving reserve_hint (reserve_hintable_range? range_with_reserve_hint?), we should rename the function instead.</p>		
FR-025 -246		25.7.18.2	te		P2846R6 which added a reserve_hint. did not cover the full set of new ranges added in C++26. This creates needless inconsistencies.	Add a reserve_hint function to concat_view (sum of the reserve_hint of the underlying ranges),	
US-153- 247		25.7.27.1, 25.7.28.1	02.1	te	The result of adjacent<0> and adjacent_transform<0> should have length one greater than that of the input range, as there are N+1 separate empty substrings of a string of size N.	Provide a specialization of adjacent_view for N==0 whose iterator's state is a single underlying iterator and a bool to absorb the first increment that can always be performed. Remove the N>0 restriction from adjacent_transform_view, which would simply use that specialization if appropriate.	
DE-248		25.7.35 [range.to.inp ut]		te	<p>The view "to_input" should be renamed to "as_input".</p> <p>The current naming of "to_input" is misleading and should better be "as_input" because we do not process the range, we only mark it differently for processing.</p> <p>Remember, we currently have</p> <ul style="list-style-type: none"> - "as_constant" and "as_rvalue" for views that make elements to deal in a different fashion without processing them. - "to<>()" to process the elements. <p>We should keep that consistent. "as_input" would make clear that we do NOT process the elements.</p>	Rename the view "to_input" should to "as_input" as described in P3828	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>"to_..." should be reserved for utilities that process elements of a range.</p> <p>This also fits with the native meaning of "as" and "to" because "as" describes how we should handle something while "to" usually is meant more active.</p>		
AT9-249		25.7.8		te	There are long-standing, UB-related safety concerns regarding mutation through a multi-pass <code>filter_view</code> .	Adopt P3725 (which has already been design approved by LEWG) which adds a way to create a single-pass <code>filter_view</code> , making the safety concern mute.	
RU-250		25.7.8	Filter view [range.filter]	te	P3725R1 highlights issues with <code>`std::ranges::filter`</code> , in particular <code>`container views::filter(large) views::reverse views::as_rvalue ranges::to<std::vector>()`</code> . Those issues are highly confusing for the language users and cause UB that should be diagnosed at compile time	Consider solutions for the issues proposed in P3725	
DE-251		25.7.8.2 [range.filter.view]		te	<p>For filter views, add a new const member functions <code>begin()</code> and <code>end()</code> when it operates on an input ranges, which supports a const <code>begin()</code>.</p> <p>P3725 demonstrates how basic use of the filter view can cause severe unexpected broken code causing misbehavior and even overwriting memory of other objects. With C++26, introducing the <code>to_input</code> (or <code>as_input</code>, see other NB comment) view, we could easily provide a workaround that brings back safety when using the filter view. For this, a small extension to the filter view is necessary, which should be provided by C++26.</p> <p>In Sofia LEWG took the following vote:</p>	<p>class filter should get:</p> <p>A new const <code>begin()</code> and const <code>end()</code> member function with constraints so that it applies to pure input ranges only:</p> <pre>constexpr const_iterator begin() const requires (input_range<const V> && !forward_range<const V> && indirect_unary_predicate<const Pred&, iterator_t<const V>>); constexpr auto end() const requires (input_range<const V> && !forward_range<const V>)</pre>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat										
					<p>From “P3725R1: Filter View Extensions for Input Ranges, Rev 1” to LWG with a recommendation to apply for C++26 (if possible) as a bug fix and as a DR for C++23, C++20.</p> <table><tr><td>SF</td><td>F</td><td>N</td><td>A</td><td>SA</td></tr><tr><td>17</td><td>10</td><td>2</td><td>0</td><td>0</td></tr></table>	SF	F	N	A	SA	17	10	2	0	0	<p>For this a corresponding new type member <code>const_iterator</code> is also introduced for filter views.</p> <p>The description of the filter view does not have to be change because it already does handle pure input ranges as underlying ranges without performing caching, which is exactly the behavior we need here to fix the severe functional failures of the current filter view.</p> <p>The exact wording will come as new version of P3725 or a separate paper.</p>	
SF	F	N	A	SA													
17	10	2	0	0													
US 154-252		26.2	11	te	“the semantics of <code>s - i</code> has” is not grammatically correct. Additionally, “type, value, and value category” are properties of expressions, not “semantics”.	Strike “the semantics of”.											
US 155-253		26.3.2	1	te	“subsumes” does not work here because <code>regular_invocable</code> and <code>invocable</code> subsume each other.	Say that the type is required to model <code>regular_invocable</code> .											
US 156-254		26.3.5		ed	The title seems outdated.	Change to “Parallel algorithm overloads”.											
US 157-255		26.4		te	<code>stable_sort</code> , <code>stable_partition</code> and <code>inplace_merge</code> range overloads should be marked <code>// hosted</code> , not <code>// freestanding-deleted</code>	Modify accordingly.											
US 158-256		26.6.9		ed	The title suggest an unwarranted parallel with “find last”.	Consider renaming the title to “Find first of” (and update the comment in the header synopsis).											

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
GB09 -257		26.7		te	Make user-defined constructors of view iterators/sentinels private Paper P3059 was not pursued, but seems like an important improvement to the design. Users should not be constructing iterators and sentinels manually, they should only obtain them by calling functions like <code>begin()</code> and <code>end()</code> .	Adopt P3059	
US 161- 258		26.7.11 26.8.2.3 26.8.3 26.8.6	6, 18 7 7 14	te	These do not handle sized-but-not-sized-sentinel ranges correctly.	Use <code>ranges::begin(r) + ranges::size(r)</code> instead of <code>ranges::end(r)</code> .	
US 159- 259		26.7.5		te	The default template argument for the type of the new value in <code>range::replace</code> and <code>ranges::replace_if</code> should not have projections applied.	Change to <code>iter_value_t<I></code> or <code>range_value_t<R></code> as appropriate.	
US 160- 260		26.7.9	10	te	This wording does not allow the source range to be empty.	Add “if first != last” or similar to paragraph 10.	
US 162- 261		26.8.5	21	te	The wording is unclear what happens if there is not such element.	Clarify that <code>out_true/out_false</code> is returned in this case.	
US 163- 262		26.8.6	1, 4	ed	Bullets 1.3 and 1.4 and paragraph 3 should say <code>E(e1, e2)</code> instead of <code>E(e1, e1)</code> .	Modify accordingly.	
US 164- 263		26.8.7.3 26.8.7.4 26.8.7.5 26.8.7.6		te	The definition of M is not clear that it considers the handling for multiple equivalent elements. The Effects paragraph does not say which elements are copied if $M > N$.	Respecify these to correct the issues, possibly by defining the resulting sequence first without regard to output capacity, and then say that the first N elements of that sequence are copied.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>The Returns paragraph is unclear for the case where no element was copied or skipped from a given range.</p> <p>The Remarks paragraph does not account for the case where elements were not copied because the output range is full.</p>		
US 165- 264		26.8.7.5	4	te	"and [first2, last2), respectively" is extraneous.	Strike It and change "elements" to "element" and "positions" to "the position".	
FR- 026 -265		26.10.17		te	The names add_sat, sub_sat, mul_sat etc are rather confusing and nondescript	<p>Rename to saturating_add, saturating_sub, saturating_mul, saturating_div</p> <p>OR</p> <p>Rename to add_saturate / sub_saturate / mul_saturate / div_saturate</p>	
US 166- 266		26.11.1		te	uninitialized_relocate is an essential algorithm for using trivial relocation that is missing from the specification.	<p>Adopt P3516.</p>  <p>p3516r2.html</p>	
FR- 027 -267		26.11.7		te	<p>P2248R8 defaults some template parameters in the algorithms to allow list initialization. P3217R0 adds it to ranges::find_last, which was forgotten.</p> <p>However, we also forgot about uninitialized_fill.</p>	Adopt P3787R0 - Adjoints to "Enabling list-initialization for algorithms": uninitialized_fill	
RU- 268		27.4.3.7	Modifiers [string.modifiers]	te	Construction of temporary string in basic_string::append/assign should be avoided for efficiency	Consider applying the fix from library issue 3662.	
GB10 -269		28.5		te	<p>Make std::format usable in constant expressions</p> <p>Throwing exceptions in constant expressions and using user-defined strings in static_assert</p>	Make std::format and integer formatters usable in constant expressions.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					are both hobbled by not being able to use <code>std::format</code> . We can already use <code>to_chars</code> in constant expressions, which would make it possible to support strings and integers in compile-time <code>std::format</code> calls. Waiting until C++29 to be able to do this would be disappointing.		
US 167- 270		28.5		te	A lot of work in the C++26 timeframe has gone into both being able to support <code>constexpr std::format()</code> and then also strongly motivating its existence (<code>static_assert</code> with user-defined messages and <code>constexpr</code> exceptions). The only thing missing is simply allowing <code>std::format()</code> to be <code>constexpr</code> . Delaying making <code>std::format</code> <code>constexpr</code> pushes the burden onto users to figure out how to do compile-time formatting themselves, despite having just adopted a standardized formatting mechanism.	Adopt P3391.  p3391r1.html	
FR- 028 -271		28.5		te	A lot of reflection use cases rely on the ability to produce strings at compile time. <code>std::format</code> is the obvious tool for such manipulation, however <code>std::format</code> was not made <code>constexpr</code> in time for C++26. We feel that this omission is an impediment to the usability of reflection features in C++26.	Adopt P3391 in C++26 (which is currently in LWG)	
RU- 272		29.5.3.1	General requirement s [rand.req.ge nl]	te	Instantiating <code>uniform_int_distribution<uint8_t></code> should either be permitted or rejected at compile time as the UB is easily detectable.	Consider applying one of the fixes from library issue 4109.	
FR- 029 -273		29.9		ge	Special math functions were added in the standard with the justification they were hard to implement and the expectation was that standard libraries maintainers would be the most qualified to do that work for the benefit of a small number of C++ users.	Reconsider whether the C++ is the appropriate vehicle for domain specific components such as linear algebra. Remove <code>[linalg]</code> from C++26. Alternatively, find a shipping vehicle that would not require an implementation on platforms not targeted by HPC platforms and other BLAS users	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>But because standard library maintainers are not domain experts in all domains, and because their time is limited and their resources scarce, multiple implementations now depend on boost to implement special math functions.</p> <p>We are concerned that <linalg> which is very clearly useful to a small subset of C++ users is going to suffer the same fate. This is aggravated by the rather large size of C++26 and the dwindling contributions to standard libraries.</p> <p>Lower level tools such as extended floating point types, mdarray and mdspan should be enough to allow interoperability in the domain and industries that do benefit from linear algebra facilities.</p>		
US 171- 274		29.9.13.9, 29.9.13.12		te	<p>As LWG Issue 4315 explains, the Returns clauses of <code>vector_two_norm</code> (29.9.13.9 [linalg.algs.blas1.nrm2]) and <code>matrix_frob_norm</code> (29.9.13.12 [linalg.algs.blas1.matfrobnorm]) say that the functions return the “square root” of an expression. However, the wording does not explain how to compute the square root. There are at least three concerns.</p> <ol style="list-style-type: none"> 1. The input <code>mdspan</code>’s <code>value_type</code> and the initial value type <code>Scalar</code> (if applicable) are not constrained in a way that would ensure that calling <code>std::sqrt</code> of that expression would be well formed. 2. The wording does not explain how to find <code>sqrt</code> via argument-dependent lookup, even though 29 [linalg] has provisions to find <code>abs</code>, <code>conj</code>, <code>real</code>, and <code>imag</code> via argument-dependent lookup. There is no “<code>sqrt-if-needed</code>” analog to <code>abs-if-needed</code>, <code>conj-if-needed</code>, <code>real-if-needed</code>, and <code>imag-if-needed</code>. 	Adopt the Suggested Fix of LWG Issue 4315 (“Insufficient specification of <code>vector_two_norm</code> and <code>matrix_frob_norm</code> ”), by constraining the input <code>mdspan</code> ’s <code>value_type</code> and <code>Scalar</code> (if applicable) to be floating-point numbers or specializations of <code>std::complex</code> .	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>3. If the <code>vector_two_norm</code> and <code>matrix_frob_norm</code> functions use <code>std::sqrt</code> to compute square roots, and if <code>Scalar</code> and the input <code>mdspan</code>'s <code>value_type</code> are both integer types, then the square root computed via <code>std::sqrt</code> would return double, but the functions would silently force a rounding conversion back to the integer type.</p> <p>Fixing this for general linear algebra value types would involve two steps.</p> <p>1. Define an exposition-only function <code>sqrt-if-needed</code> that uses <code>std::sqrt</code> for arithmetic types, and finds <code>sqrt</code> via argument-dependent lookup otherwise.</p> <p>2. Change the <code>vector_two_norm</code> and <code>matrix_frob_norm</code> wording so the return type of the functions that take a <code>Scalar</code> <code>init</code> parameter is <code>decltype(sqrt-if-needed(init + a * a))</code> instead of <code>decltype(init + a * a)</code>.</p> <p>The second part of this could be considered a design change, though, so for now we propose simply constraining the function so that <code>Scalar</code> and the <code>mdspan</code>'s <code>value_type</code> are either floating-point numbers or <code>std::complex</code>. This will permit a backwards-compatible fix later.</p> <p>There is some debate whether a Constraint would permit a backwards-compatible fix. If WG21 does not think it would, then a Mandates would be the right option.</p>		
US 172- 275		29.9.14, 29.9.15		te	As LWG Issue 4137 explains, some of the Mandates, Preconditions, and Complexity elements of some BLAS 2 and BLAS 3 algorithms in [linalg] are incorrect.	Adopt the Proposed Resolution of LWG Issue 4137 ("Fix Mandates, Preconditions, and Complexity elements of [linalg] algorithms").	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 169- 276		29.9.2, 29.9.13.8		te	As LWG Issue 4302 explains, it is impossible to implement <code>vector_sum_of_squares</code> in a way that satisfies both requirements on <code>result.scaling_factor</code> , without imposing more requirements on <code>InVec::value_type</code> and <code>Scalar</code> (if applicable). There are other concerns explained in the Issue.	Adopt the Proposed Resolution of LWG Issue 4302 ("Problematic <code>vector_sum_of_squares</code> wording"), by removing all mentions, declarations, and descriptions of <code>vector_sum_of_squares</code> from <code>[linalg]</code> .	
US 168- 277		29.9.2, 29.9.4.1, 29.9.14.6, 29.9.14.7, 29.9.14.8, 29.9.15.4, 29.9.15.5		te	As P3371 explains, <code>[linalg]</code> 's rank-1, rank-2, rank-k, and rank-2k update functions have behavior not consistent with the Basic Linear Algebra Subroutines (BLAS). Changing their behavior would be a breaking semantic change to C++26 with no syntactic indication. P3371, which proposes a fix, has passed LEWG review and awaits LWG review. The implementation of P3371 has been merged into the main reference <code>std::linalg</code> implementation.  p3371r4.html	Adopt the latest revision of P3371 (currently R4), "Fix C++26 by making the rank-1, rank-2, rank-k, and rank-2k updates consistent with the BLAS." Please note that the changes proposed in P3371R4 are rebased atop the changes proposed in LWG Issue 4137, "Fix Mandates, Preconditions, and Complexity elements of <code>[linalg]</code> algorithms." Thus, applying the changes in P3371R4 should resolve LWG Issue 4137.  p3371r4.html	
US 170- 278		29.9.3	4	te	As LWG Issue 4136 explains, the Hermitian functions in <code>[linalg]</code> currently have undefined behavior if they encounter a diagonal matrix element with nonzero imaginary part. The Basic Linear Algebra Subroutines (BLAS) define this behavior by only using the real part of the diagonal elements. Given that <code>[linalg]</code> aims to follow BLAS behavior and aims also to be able to use existing BLAS libraries where possible, it would be best for <code>[linalg]</code> to adopt BLAS behavior for diagonal matrix elements.	Adopt the Proposed Resolution of LWG Issue 4136 ("Specify behavior of <code>[linalg]</code> Hermitian algorithms on diagonal with nonzero imaginary part"), by specifying that functions whose name starts with "hermitian" will use <code>real-if-needed(m[i, i])</code> to access diagonal elements <code>m[i, i]</code> .	
AT8- 279		29.10.3		te	The <code>simd</code> library is missing reduction overloads (<code>reduce</code> , <code>reduce_min</code> , <code>reduce_max</code>) for non- <code>simd</code> , <code>vectorizable</code> types.	Add the missing overloads by applying P3690 (which has already been design approved by LEWG).	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 176- 280		29.10.3		te	It does not appear that any of the class templates in <simd> can be profitably specialized by the user – or that the non-member functions can be used with such user specializations.	Disallow all such specializations. Then remove 29.10.4 paragraph 3 as redundant.	
US 175- 281		29.10.3		ed	The ordering of the declarations in the synopsis does not match the order of subclauses.	Reorder the declarations and/or the subclauses as appropriate so that the content appear in a consistent, logical order.	
US 174- 282		29.10.3		te	zero_element and uninit_element should be inline and not static	Modify accordingly.	
US 173- 283		29.10.3		ed	Last overload of compress in the synopsis should say M::value_type and not V.	Modify accordingly.	
US 177- 284		29.10.4		ed	The title of this subclause is “vec type traits” but the traits defined apply to masks too.	Rename the title, possibly to “Data-parallel type traits”.	
DE- 285		29.10.7.2	p13 & p14	te	convertible_to<array<string, 4>, vec<float, 4>> is true, but it should really be false. Likewise for convertible_to<array<double, 4>, vec<float, 4>> or convertible_to<array<complex<float>, 4>, vec<float, 4>>.	What is currently under Mandates should go into Constraints.	
DE- 286		29.10.7.2	p1–4	te	The broadcast constructor in the TS allowed construction from (unsigned) int, allowing e.g. 'vec<float>() + 1', which is ill-formed in the CD (breaking existing code). The design intent behind std::simd was that this works. But the understanding was that because of language	Add a 'consteval' broadcast constructor overload and adjust the existing broadcast constructor if necessary. Consider P3844R0. At least the design space must be considered to ensure C++26 does not preclude a solution.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					limitations (missing constexpr function arguments) this wasn't possible. There is new information.		
DE- 287		29.10.7.2	p17 & p18	te	This deduction guide does not suffice to make basic_vec x = mask<float>(); work. If the comment on 29.10.9.4 gets resolved then there is no easy way (without using decltype) for users to spell the basic_vec specialization that the basic_mask type prefers to convert to. Using 'auto' instead of 'basic_vec' above does not work, since it would not invoke the conversion from basic_mask to basic_vec.	Add a deduction guide: template <size_t Bytes, class Abi> basic_vec(basic_mask<Bytes, Abi>) -> see below; Then use the equivalent wording that is used to resolve 29.10.9.4 to determine the basic_vec specialization.	
DE- 288		29.10.7.2	p5	te	This constructor is missing a constraint. Consider conversion from vec<complex<float>, 4> to vec<float, 4>.	Add "and constructible_from<value_type, U> is true"	
US 178- 289		29.10.8		ed	The subclause titles sometimes use "vec" and sometimes use "basic_vec".	Make them consistent and update the synopsis as appropriate.	
US 183- 290		29.10.8.10	5	te	The order of the index positions in set-indices needs to be specified.	Add "in ascending order" to bullet 5.1.	
US 184- 291		29.10.8.11		ed	This is finally just "vec" but the title says "simd memory permute".	Correct the title.	
RO 3- 292	1-10	29.10.8.3		te	The operator== for std::simd::basic_simd breaks the Regular semantic expectations. As noted in http://wg21.link/P2892R0 , simd types are not Regular, which is inconsistent with the rest of the standard library's design.	simd meets the semantic requirements of Regular types, including operator== returning a bool for scalar comparisons. Alternative resolutions:	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>Specifically, an expression such as:</p> <pre>if (vec1 == vec2) { /* ... */ }</pre> <p>will not compile, even though operator== is provided. This behaviour is surprising, violates the principle of least astonishment, and creates an inconsistency in the interface: the operator exists, yet cannot be used in a context where a boolean value is expected.</p> <p>This undermines interoperability with generic code that assumes regularity and boolean comparability for types that provide equality operators.</p>	<ul style="list-style-type: none"> simd adopts a distinct naming or API pattern to avoid presenting an equality operator that cannot be used in normal boolean contexts (e.g., equals()). 	
US 179- 293		29.10.8.4		ed	These are member functions, not non-member functions.	Move this subclause under 20.10.7.	
US 181- 294		29.10.8.8 29.10.8.9 29.10.8.10		ed	The title says “vec”, and these are placed under a subclause titled “basic_vec non-member operations”, but these functions do not apply to just “vec”s; nor do they match the header synopsis.	Make things consistent, possibly by reorganizing as appropriate.	
US 180- 295		29.10.8.8 29.10.8.9 29.10.8.10		ed	Various places uses “V” when “V or M” is meant.	Either rename M to V or reword.	
US 182- 296		29.10.8.8	2	te	This need to clarify that we are not actually evaluating the expression (which might be ill-formed) but are doing concepts satisfaction checking.	Use “is satisfied” instead of “is true”.	
DE- 297		29.10.9.4		te	operator+, operator-, and operator~ cannot be implemented for mask<complex<double>>.	Resolve LWG4238	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
DE- 298		29.10.9.4		te	The use of basic_mask's Abi tag for the return type is wrong for mask<complex<T>> and mask<complex<floating-point>> for targets like Intel's IvyBridge.	Resolve the corresponding LWG issue: spell out constraints on the return type rather than spelling out the exact ABI tag type.	
US 185- 299		29.10.9.5 29.10.9.6		ed	There is no such thing as "conversion operator".	Combine the two subclauses into one titled "basic_mask conversions". Update the class definition accordingly.	
US 186- 300		29.10.10		ed	The title "Non-member operations" is too general.	Add "basic_mask".	
US 187- 301		29.10.10.03		ed	The parameters need to be named as they are referenced in paragraph 2.	Name them lhs and rhs.	
GB11 -302		29.11.1		te	Add feature test macro for stdckdint.h The <code>__STDC_VERSION__</code> macro defined by C is not sufficient to check if a C++26 implementation provides a usable <code><stdckdint.h></code> header. If a libc header of that name is found somewhere in the C++ compiler's include paths (e.g. in <code>/usr/include</code>) then it might be included and would define the macro. However, the libc version of the header might use C-specific features such as <code>_Bool</code> or <code>_Generic</code> and cause errors in a C++ program. A C++-specific macro would only be present in a C++-aware header, and so a C++ program could check that to be sure that the header is usable in C++ code.	Define a new <code>__cpp_lib_stdckdint_h</code> macro.	
US 188- 303		31.6.3.5.5	04.1	ed	"effects" is the wrong word here.	Use "affects".	

1 MB = Member body / NC = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 Type of comment: ge = general te = technical ed = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 189- 304		31.12.6.1, 31.12.6.5.6, 31.12.6.5.7, D.22.2		ge	<p>The <code>system_encoded_string()</code> and <code>generic_system_encoded_string()</code> member functions of <code>std::filesystem::path</code> are misnamed.</p> <p>These functions were added as renamed versions of <code>string()</code> and <code>generic_string()</code> respectively by P2319R5 (Prevent path presentation problems). The original function names are now deprecated.</p> <p>The C++ standard does not define “system encoding” and casual use of such a term is at best ambiguous. In common language, one might expect the “system encoding” to correspond to the environment encoding (for which <code>std::text_encoding::environment()</code> provides a definition) or the encoding of the current locale. However, neither of these encodings corresponds to the encoding these functions are expected to use.</p> <p>31.12.6.3.2 ([fs.path.type.cvt]) defines “native encoding” and, for <code>char</code>-based filesystem paths, states (in a note) that “For Windows-based operating systems, the native ordinary encoding is determined by calling a Windows API function.” It is not specified which Windows API function is consulted, but existing implementations call <code>AreFileApisANSI()</code> to choose between the Active Code Page (CP ACP) and the OEM Code Page (CP OEMCP). Microsoft’s implementation also checks for a thread-local locale and, if the locale encoding is UTF-8, prefers that (CP UTF8) over either of the other two.</p>	<p>Do one of these:</p> <ol style="list-style-type: none"> 1) Remove <code>system_encoded_string()</code> and <code>generic_system_encoded_string()</code> and <code>undeprecate string()</code> and <code>generic_string()</code>. 2) Rename <code>system_encoded_string()</code> and <code>generic_system_encoded_string()</code> to names that reflect the defined “native encoding”. For example, <code>native_filesystem_string()</code> and <code>generic_native_filesystem_string()</code>. Note however that “native” is also used to define an operating system dependent path format and that a <code>native()</code> member function already exists. 3) Rename “native encoding” to “filesystem encoding” and rename <code>system_encoded_string()</code> and <code>generic_system_encoded_string()</code> accordingly. For example, to <code>filesystem_encoded_string()</code> and <code>generic_filesystem_encoded_string()</code>. <p>Replace the note in 31.12.6.3.2 ([fs.path.type.cvt]) with normative specification for the encoding to be used (with explicit mention of <code>AreFileApisANSI()</code> and any other Windows APIs required to identify the intended encoding).</p>	
US 190- 305		31.12.6.5.6 31.12.6.5.7	9 7	ed	<p>These should say <code>std::format</code> as we would otherwise find <code>path::format</code>.</p>	Add “std::”.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
GB14 -306		32.5.12		te	Remove <code>stdatomic.h</code> from C++ The goal of the C++ version of the <code>stdatomic.h</code> header was to provide a portable, cross-language way to refer to atomics. However, simply adding some macros that make the same names valid in C and C++ does nothing to address any incompatibilities between the definitions of <code>_Atomic(T)</code> and <code>std::atomic<T></code> . There is no guarantee of ABI compatibility, and pretending otherwise makes subtle ODR violations more likely.	Remove <code>stdatomic.h</code>	
US 191- 307		32.5.2		ed	This synopsis can benefit from the new freestanding specification style.	mark the header <code>//</code> mostly freestanding, then remove all the <code>//</code> freestanding comments and add <code>//</code> hosted for <code>atomic_signed_lock_free</code> and <code>atomic_unsigned_lock_free</code> .	
GB12 -308		32.5.6		te	<code>atomic wait/notify</code> should not be required for freestanding Requiring atomic waiting functions to be supported for freestanding implementations adds unreasonable burden on implementations. For some implementations there is a freestanding subset which can be supported by simply disabling parts of a hosted implementation. But to support atomic waiting functions for freestanding requires a completely separate implementation, probably based on spinlocks. For a typical implementation that already supports two implementations of those functions (one using OS primitives like <code>futexes</code> or similar kernel features, and one using <code>mutexes</code> and <code>condition variables</code>), including those functions in the freestanding subset requires supporting a third spinlock-based (and so very low quality) implementation. If users want a spinlock, they can easily make one themselves, and it seems unlikely anybody would expect <code>atomic::wait</code> to be a low quality implementation.	Remove atomic waiting functions from the freestanding subset (it would still be permitted in a freestanding implementation, but not required).	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
GB13 -309		32.5.7		te	<p><code>atomic_ref<T></code> is not convertible to <code>atomic_ref<const T></code></p> <p>The following paper added cv qualifiers to <code>atomic</code> and <code>atomic_ref</code>: P3321R1 "cv-qualified types in <code>atomic</code> and <code>atomic_ref</code>"</p> <p>https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3323r1.html</p> <p>However, the conversion constructor between cv qualifiers is overlooked.in <code>[atomics.ref.generic]</code> hence <code>atomic_ref<T></code> is not convertible to <code>atomic_ref<const T></code>.</p> <p>There is no reason why the conversion should fail. This is a valid use case where one thread only needs the load operations.</p>		
FR-030 -310		32.5.7.2		te	<p>P2835R7 added an <code>address()</code> method that allows accessing the underlying pointer of an <code>atomic_ref</code>. This is useful for comparisons, however accessing the underlying object through that address method is generally undefined behavior.</p> <p>Given that the main reason for the current design is that <code>intptr_t</code> is not mandated to exist in the standard, we suggest to make <code>(u)intptr_t</code> mandatory in C++26 - which was already on track for C++29</p>	<p>adopt P3248 R4 Require <code>[u]intptr_t</code> in C++26</p> <p>modify <code>std::atomic_ref::address</code> to return <code>uintptr_t</code></p>	
US 193-311		32.5.7.3 32.5.7.4 32.5.7.5	10 10 11	te	<p>These should have a constraint that the relevant type is not <code>const</code>.</p>	Add appropriate constraints.	
US 192-312		32.5.7.3 32.5.7.4 32.5.7.5	2 2 2	ed	<p>These should say <i>integral-type</i>, <i>floating-point-type</i>, and <i>pointer-type</i> respectively rather than <code>T</code>.</p>	Modify accordingly.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 195- 313		32.5.7.5	11	te	This should say “remove_pointer_t<pointer-type>”, not T.	Modify accordingly.	
US 194- 314		32.5.7.5	2	ed	This should be a full specialization rather than a partial one.	Strike “class T”. Also delete “partial” from the subclause title.	
US 196- 315		32.5.8.3 32.5.8.4	5, 14 5, 15	ed	These should say <i>integral-type</i> and <i>floating-point-type</i> respectively rather than T.	Modify accordingly.	
US 197- 316		32.5.8.4		ed	Some functions (fetch_max*) are not sufficiently indented. Additionally, the second line of function declarations are indented inconsistently.	Make indentation consistent.	
US 198- 317		32.5.8.5	4	ed	These should say <i>see above</i> instead of ptrdiff_t.	Modify accordingly.	
CA- 318		33		te	Operation states are required to store an actual receiver which is required to contain at least a pointer in order to report completion including in situations where the offset of the parent operation state is statically known. This provides an overhead of at least the size of a pointer at each level of asynchronous operation composition.	Adopt P3425, “Reducing operation-state sizes for subobject child operations.”	
FR- 031 -319		33		te	P3826R0 proposes removing the ability to remove the customization of the sender algorithms. When adopted, this makes them unusable for parallel use cases like GPUs. Having uncustomizable sender algorithms is still useful for concurrency use cases, but having them means we’re constrained in how we can add proper customization in C++29. When adopting P3826R0 we should therefore go one	When adopting P3826R0, also make the following changes <ul style="list-style-type: none"> Remove everything in [exec.adapt] Remove everything in [exec.affine.on] As a consequence of the removal, references to removed algorithms need to be updated (e.g. behavior	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					<p>step further and also defers the sender adaptors to C++29.</p> <p>Additionally, it buys us time to figure out a solution for P3425 - Reducing operation-state sizes for subobject child operations.</p> <p>Note that even if the algorithms are deferred to C++29, we can still accomplish all the goals for senders/receivers:</p> <ul style="list-style-type: none"> We have the concepts, customization points, and vocabulary to talk about senders, enabling third-party library and unblocking work on networking. With <code>std::execution::task</code>, we have a solution that makes coroutines usable. We still have the parallel scheduler to get access to the default execution context and the execution scopes for structured concurrency. <p>We can still compose senders using coroutines or by third-party libraries that implement sender algorithms.</p>	<p>specified in terms of <code>execution::affine_on</code> or <code>execution::write_env</code>). This can be avoided by making them exposition only instead.</p>	
US 205-320		33.4 33.15, 33.16, 33.16.1, 33.16.2, 33.16.3		ge	<p>The <code>system_context_replaceability</code> name for a namespace is misleading and unprecedentedly long for a namespace.</p> <p>The mentioned name was chosen when the main API of P2079R10 (Parallel scheduler) was called <code>system_scheduler</code> in earlier revisions of the paper. In the current C++ working draft the main API is called <code>parallel_scheduler</code>. There is no other mentioning of <code>system_context</code> beyond this namespace name.</p> <p>Also, this name is the longest among all the namespace names the C++ working draft has.</p>	<p>Do one of these (rename stable names accordingly):</p> <ol style="list-style-type: none"> Remove <code>std::execution::system_context_replaceability namespace</code>. Rename <code>std::execution::system_context_replaceability</code> to: <ol style="list-style-type: none"> <code>std::execution::replacement</code>, or <code>std::execution::replacement_functions</code>, or <code>std::execution::psr</code> (abbreviation from <code>parallel_scheduler_replacement</code>), or 	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					 p2079r10.html	d. something else meaningful.	
US 204- 321		33.4 33.15		ed	It seems more stylistically consistent to have the definition of parallel_scheduler in 33.15 before the text.	Remove { unspecified } from the declaration of parallel_scheduler in the synopsis. Add the definition to the beginning of 33.15	
US 201- 322		33.4		ed	The std::execution blocks at the end should be merged, possibly with system_context_replaceability nested. The task related items should be moved after with_awaitable_senders to match subclause order.	Modify accordingly.	
US 200- 323		33.4		ed	Some comments are bare cross references without text	Add some descriptive text.	
US 199- 324		33.4		te	enable_sender is not in the header synopsis.	Add it.	
US 206- 325		33.4	2	te	The “sizeof...(Env) > 1 is true” part seems unreachable because CS is ill-formed in that case.	Remove it.	
US 202- 326		33.4, 33.5.5, 33.5.6, 33.9.2, 33.9.12.3, 33.9.12.4, 33.9.12.5, 33.9.12.6, 33.9.12.8, 33.9.13.1,		te	The current approach to domain customization hinders GPU scheduler customization, which was a major motivation for std::execution's design. P3718 solves this, based on experience writing two separate GPU-based std::execution libraries. The solution will benefit all schedulers that depend on customization for performance. Fixing this issue later would be a breaking change to C++26.	Adopt the latest revision of P3718 (currently R0), “Fixing Lazy Sender Algorithm Customization, Again.”  p3718r0.html	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)


2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
		33.9.13.2			 p3718r0.html		
US 203- 327		33.4, 33.9.12.11, 33.15, 33.16.2, 33.16.3		te	<p>The Working Draft currently provides only three sender adapters for parallel execution of a function over a [0, N) integer range: bulk, bulk_chunked, and bulk_unchunked. Of these, bulk and bulk_chunked behave most like a “parallel for loop” and are thus the sender adapters for most users, but they have obscure names that do not suggest “parallel for loop.” The bulk_unchunked sender adapter may give unexpectedly poor performance if users do not understand that it is a special-purpose function that may create more execution agents than users might realize. Thus, it would be best to rename bulk and bulk_chunked so that users looking for a “parallel for loop” can find them, by including “for_loop” in their names. The less used bulk_unchunked can thus be renamed to “bulk,” which better conveys the original intent.</p>	<p>Change 33.4 [execution.syn] as follows.</p> <ul style="list-style-type: none"> Replace all instances of bulk_t (the tag type name) with for_loop_t Replace all instances of bulk (the inline constexpr variable of type bulk_t) with for_loop Replace all instances of bulk_chunked_t (the tag type name) with for_loop_chunked_t Replace all instances of bulk_chunked (the inline constexpr variable of type bulk_chunked_t) with for_loop_chunked Replace all instances of bulk_unchunked_t (the tag type name) with bulk_t Replace all instances of bulk_unchunked (the inline constexpr variable of type bulk_unchunked_t) with bulk <p>Rename 33.9.12.11 [exec.bulk] title as follows: 33.9.12.11 execution::bulkfor_loop, execution::bulkfor_loop_chunked, and execution::bulk_unchunked [exec.bulk]</p> <p>Change 33.9.12.11 [exec.bulk] by replacing names as in 33.4.</p> <p>Change 33.15 [exec.par.scheduler] and 33.16 [exec.sysctxrep] by replacing names as in 33.4 and by making the following changes.</p> <ul style="list-style-type: none"> Replace all instances of the term bulk chunked proxy ([exec.par.scheduler] 	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)




² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						<p>paragraphs 7 and 10.1.1) with the term chunked for loop proxy.</p> <ul style="list-style-type: none"> Replace all instances of the term bulk unchunked proxy ([exec.par.scheduler] paragraphs 8 and 11.1.1) with the term bulk proxy. Replace all instances of schedule_bulk_chunked ([exec.par.scheduler] 10.1, [exec.sysctxrepl.psb] 1, 4) with schedule_for_loop_chunked. Replace all instances of schedule_bulk_unchunked ([exec.par.scheduler] 11.1, [exec.sysctxrepl.psb] 1, 7) with schedule_bulk. 	
US 207- 328		33.5.5, 33.5.6, 33.9, 33.13.3, 33.15		te	<p>Section [exec] in the Working Draft has sender algorithms that are customizable. The customization mechanism has seen a fair bit of recent churn. P3718 was the latest effort to shore up the mechanism. Unfortunately, there are gaps in its proposed resolution. See D3826R0 (https://isocpp.org/files/papers/D3826R0.html) for technical details.</p> <p>It is too late for additional fixes. The ability to customize sender algorithms should be removed for C++26 and added back once the issues have been fixed.</p> <div>   </div> <p>d3826r0.html p3718r0.html</p>	<p>Accept the proposed resolution in D3826R0 (https://isocpp.org/files/papers/D3826R0.html), which will become https://isocpp.org/files/papers/P3826R0.html once published).</p> <div>  </div> <p>d3826r0.html</p>	
CA- 329		33.7.1		te	<p>Receivers are permitted to throw from their move constructor which necessitates pessimization as described in P3388.</p>	<p>Adopt P3388, “When Do You Know `connect` Doesn’t Throw?”</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FR-032 -330		33.7.1		te	Adopt P3388R2 - When Do You Know connect Doesn't Throw? - as a necessary bugfix for C++26. It was already forwarded by LEWG.	Adopt P3388R2.	
FI-331		33.9		te	The sender algorithm customization mechanism isn't quite functional, based on practical implementation and deployment experience. It will need more work to get right, and the time for such getting-right is in C++29. We can keep the rest of Senders&Receivers just fine, the application-level API is unaffected by this.	Adopt the paper P3826.	
US-209-332		33.9		te	<i>unspecified-exception</i> is underspecified.	Specify that it is publicly and unambiguously derived from exception, and not related to dependent_sender_error.	
US-208-333		33.9		ed	Various cross-references for impls-for should go to 33.9.2 rather than 33.9.1.	Update cross-references.	
CA-334		33.9.10	Paragraph 6	te	The fact that `connect` is allowed to throw for same receivers with a certain environment type but not for other receivers with that same environment type necessitates pessimization as described in P3388.	Adopt P3388, "When Do You Know `connect` Doesn't Throw?"	
FR-033 -335		33.9.12		ed	The headings for [exec.associate], [exec.stop.when], [exec.spawn.future], [exec.spawn] use std::execution::foo, the rest of the clause only execution::foo.	Remove the redundant std:: in the specified headings.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FR-034 -336		33.9.12		te	<p>The naming of the monadic operations in <code>std::execution</code> (<code>then</code>, <code>let_value</code>, <code>let_error</code>, <code>let_stopped</code>) are inconsistent with the naming of the monadic operations elsewhere in the standard library (<code>transform</code>, <code>and_then</code>, <code>or_else</code>). They should be renamed for greater consistency.</p> <p>See P3845R0 for more motivation.</p>	<ul style="list-style-type: none"> Rename <code>std::execution::then</code> to <code>std::execution::transform</code> Rename <code>std::execution::let_value</code> to <code>std::execution::and_then</code> Rename <code>std::execution::let_error</code> to <code>std::execution::or_else_error</code> Rename <code>std::execution::let_stopped</code> to <code>std::execution::or_else_stopped</code> 	
FR-035 -337		33.9.12		te	<p>The use of "counting" in <code>std::execution::simple_counting_scope</code> and <code>std::execution::counting_scope</code> is both non-descriptive – all possible implementations of async scopes have some implicit/explicit count to drive their internal state machine - and not useful to the user. We should remove the specifier.</p> <p>Are we also sure that we want to imply that the scope with the stop source is the reasonable default that defers the shorter name?</p>	<p>Rename <code>std::execution::{simple_counting_scope, counting_scope}</code> either to</p> <ul style="list-style-type: none"> <code>std::execution::{simple_scope, scope}</code> if we want to imply that the scope with stop source is the reasonable default, or <code>std::execution::{scope, stoppable_scope}</code> if we want to imply that the scope without stop source is the default, or <p><code>std::execution::{simple_scope, stoppable_scope}</code> if we want to imply that neither scope is the default.</p>	
CA-338		33.9.12.10		te	<p>As described in detail in P3373 <code>`let_value`</code>, <code>`let_error`</code>, and <code>`let_stopped`</code> all extend the lifetime of the predecessor operation state thereby unnecessarily increasing the lifetime of those operation states and the storage occupied by a <code>`let_value`</code>, <code>`let_error`</code>, or <code>`let_stopped`</code> operation state.</p>	<p>Adopt P3373, "Of Operation States and Their Lifetimes," thereby causing <code>`let_value`</code>, <code>`let_error`</code>, and <code>`let_stopped`</code> to function in the way the implementations in libunifex do with respect to the predecessor operation state's lifetime.</p>	
US-221-339		33.9.12.11	5-7	te	<p><code>check-types</code> was added to <code>impls-for<bulk_chunked_t></code>'s definition, but the specification is under <code>impls-for<bulk_unchunked_t></code>. Additionally, the specification of <code>check-types</code> is incorrect; <code>data-type<Sndr></code> is the product-type tuple, not an invocable.</p>	<p>Add a <code>check-types</code> to <code>impls-for<bulk_unchunked_t></code> as well. Specify it appropriately. Change the definition of <code>check-types</code> to extract the invocable and shape types, and check invocability correctly.</p>	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 222- 340		33.9.12.11	6	te	impls-for<bulk_unchunked_t>::complete unpacks state incorrectly, since it is now a three-element tuple.	Change “[shape, f]” to “[policy, shape, f]”.	
US 225- 341		33.9.12.12 33.9.12.13	18 6	te	Tuple’s constructor is not required to be conditionally noexcept. As a result, the “potentially-throwing” check is not guaranteed to work.	Either add conditional noexcept or change these to check elementwise construction.	
US 224- 342		33.9.12.12	5	te	This wording does not specify that other queries are not forwarded.	Specify that.	
US 223- 343		33.9.12.12	5	te	Bullet 5.3 should say “get_stop_token_t” instead of “stop_token_t”, which does not exist.	Modify accordingly.	
US 220- 344		33.9.12.12	5	ed	This should be made a <i>Returns</i> : paragraph. state.stop-src should be stop_src and get_env(rcvr).query should be env.query.	Modify accordingly.	
US 226- 345		33.9.12.17	2	te	The bullet 2.2 case still needs to evaluate token.	Change the last “sndr” to “(void) token, sndr, except that token and sndr are indeterminately sequenced”.	
US 227- 346		33.9.12.18 33.9.13.3		te	The allocator is rebound to <i>spawn-state</i> or <i>spawn-future-state</i> and used as a data member when the type is still incomplete. Allocators are not required to support incomplete types.	Consider rebinding the allocator at the point of use instead.	
US 229- 347		33.9.12.18	16	te	The template arguments of the type of s is unclear.	Specify the actual type.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 228- 348		33.9.12.18	6	te	ssource-t should be required to be default-constructible and the resulting object should not be disengaged. It should also be in code font.	Add wording to that effect and change the font.	
US 218- 349		33.9.12.3	3	ed	Bullet 3.3 is really specifying check-types and should be its own paragraph.	Modify accordingly..	
US 219- 350		33.9.12.9	5	te	The selection of completion signature is incorrect.	Replace set_value_t with decayed-typeof<set-cpo>.	
US 211- 351		33.9.2	24	te	The definition of Sndr in bullet 24.4 should probably decay Data and Child, consistent with the return type.	Modify accordingly.	
US 212- 352		33.9.2	26	ed	This paragraph is talking about the default template argument of make-sender but is separated far from it.	Move it before the code block and turn it into a Remarks: paragraph.	
US 213- 353		33.9.2	26-45	ed	Much of paragraphs 27-45 are far from the entities they specify and haphazardly organized.	Split the big code block before paragraph 26 into pieces and move the corresponding parts of paragraphs 27-45 under them.	
US 210- 354		33.9.2	3	te	It is unclear if MAKE-ENV is still needed when we have prop.	If not, strike it and replace its uses.	
US 214- 355		33.9.2	41	te	The fold expression can pick up overloaded comma operators.	Cast the call to get_completion_signatures to void.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 215- 356		33.9.2	43	ed	The declaration of basic-sender:: get_completion_signatures does not match the class definition.	Make it consistent.	
US 216- 357		33.9.2	49	ed	This paragraph should say "function template" instead of "function"; should say "new object of type remove_cvref_t<T>" instead of "new object of type T". Missing closing parens for the "i.e." after "is valid".	Modify accordingly.	
CA- 358		33.9.6		te	There have been many late-breaking issues discovered with sender customization, we shouldn't ship it when we know it's broken in multiple ways (in addition to ways we haven't yet discovered).	Adopt P3826, "Defer Sender Algorithm Customization to Post-C++26."	
US 217- 359		33.9.9	4	ed	This paragraph is a general requirement and should not be indented.	Unindent it.	
US 230- 360		33.10	8	te	fn can be called multiple times and therefore should not be forwarded.	Remove the std::forward.	
US 231- 361		33.12.1.3 33.12.1.4		ed	<i>count</i> and <i>state</i> are not data members and the various states are not enumerators. They should not be in code font.	Either make them non-code-font, or make then actual exposition-only data members/enumerators.	
US 236- 362		33.13.3		te	The wording of affine_on doesn't have a specification for the default implementation. For other algorithms the default implementation is specified.	Provide a specification for the default implementation. See LWG4344.	
US 235- 363		33.13.3		te	The main purpose of affine_on is to make sure work resumes on a specific execution context. The scheduling operation may take some time and work may be cancelled in the meantime. If this cancellation causes the scheduling to be cancelled work cleaning up after the cancellation	Change the specification of affine_on to suppress forwarding the stop token to the scheduling operation. See LWG4332.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					would be executed on the wrong execution context. Thus, the stop token from the receiver's environment should only be forwarded when connecting the sender but not to scheduling the operation.		
US 234- 364		33.13.3		te	affine_on is specified to take a sender and a scheduler as arguments. The scheduler is meant to match the scheduler obtained from the get_scheduler query on the receiver's environment. Thus, the scheduler argument is redundant and the semantics become weird if these two schedulers don't match. The affine_on algorithm should only take the sender as argument.	Change the specification and use of affine_on to omit the scheduler parameter and use the scheduler from the receiver's environment instead. See LWG4331.	
US 233- 365		33.13.3		te	<p>The specification of affine_on uses "current execution resource" and it is unclear what that means exactly. Additionally, it is unclear what the difference between affine_on and continues_on is. The intended difference for affine_on is to avoid unnecessary scheduling which continues_on is already allowed to do in some cases, too.</p> <p>The intended semantics is that affine_on will either complete inline on whatever execution agent it was started on or it will complete asynchronously on the specified execution context. With this formulation affine_on may complete on one of two different execution context if it is started on an execution context that is different from the one specified by the scheduler.</p>	Clarify the specification of affine_on to describe the differences compared to continues_on. See LWG4330.	
US 232- 366		33.13.3		te	There are no customizations of affine_on for other algorithms specified. For example, affine_on(just(), sched) can be equivalent to just() because just() completes inline and, thus, on the correct execution context.	Specify customisations of affine_on for sender arguments which don't change the scheduler. See LWG4329.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 239- 367		33.13.5		te	As specified, there is an implicit precondition that sch_ is not moved from on all the member fucntions. If that is intended, the precondition should be made explicit.	Make the precondition explicit.	
US 238- 368		33.13.5		te	When using task_scheduler algorithm customizations are not picked up. For example, even if the task_scheduler is actually referring to a parallel_scheduler, bulk will execute work sequentially. This behavior may be surprising.	Consider forwarding algorithms in a way which picks up customisation, at least for known algorithms and/or known schedulers. See LWG4336.	
US 237- 369		33.13.5		te	The result of schedule(sched) for a scheduler sched is only required to be movable. An object of this type may need to be forwarded to an operation state constructor by task_scheduler::ts-sender::connect. Thus, this function should be qualified with an rvalue reference.	Add an rvalue reference qualifier for task_scheduler::ts-sender::connect both in the synopsis (paragraph 8) and the specification (paragraph 10). See LWG4342.	
US 240- 370		33.13.5	1	te	shared_ptr owns a pointer (or nullptr_t), not the pointee, but SCHED wants the pointee.	Say that SCHED(s) is the object pointed to by the pointer owned by s.sch_.	
US 241- 371		33.13.5	7	ed	This sentence should end with a period instead of a semicolon. Additionally, there should be a "the" before "type".	Modify accordingly.	
US 242- 372		33.13.6		te	The specification of task doesn't spell out when the coroutine frame is destroyed (i.e., when handle.destroy() is called). As a result the lifetime of arguments passed to the coroutine and/or of local variables in the coroutine body may be longer than expected.	Move the result and error types from the promise_type to the state type and requires that the coroutine frame is destroyed before invoking any of completion dispositions. See LWG4339.	
US 246- 373		33.13.6.2		te	The specification of task doesn't require symmetric transfer which can help with stack overflow. Also, when another task is co_awaited the scheduler on which the task resumes is known and can be used to avoid unnecessary scheduling by comparing the scheduler currently installed by in two tasks involved.	Specify that symmetric transfer is used when a task co_await's another task. See LWG4348.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 245- 374		33.13.6.2		te	The task type doesn't have an operator <code>co_await()</code> . Thus, a task can't be <code>co_awaited</code> directly from a coroutine. Using <code>as_awaitable</code> on the task object also doesn't work because this function requires a reference to the promise as second argument.	Consider adding an operator <code>co_await()</code> . See LWG4338.	
US 244- 375		33.13.6.2		te	Coroutines can't be copied. Thus, a task can be <code>connect()</code> just once. To represent that <code>task::connect()</code> should be rvalue reference qualified but currently it isn't.	Add an rvalue reference qualifier for <code>task::connect()</code> both in the synopsis and the specification (paragraph 3). See LWG4341.	
US 243- 376		33.13.6.2		te	The design discussion of task describes defaults for the two template parameters <code>T</code> and <code>Environment</code> of task but these defaults are not reflected in the synopsis of <code>[task.class]</code> . This is an oversight and should be fixed. The default for <code>T</code> should be <code>void</code> and the default for <code>Environment</code> should be <code>env<></code> (the design paper used <code>empty_env</code> but this struct was replaced by the class template <code>env</code> by P3325r5).	Add default template arguments for task for <code>T = void</code> and <code>Environment = env<></code> in the synopsis of <code>[task.class]</code> : <pre>namespace std::execution { template<class T = void, class Environment = env<>> class task { ... }; }.</pre> See LWG4343.	
US 247- 377		33.13.6.2	3	te	<code>completion_signatures</code> needs to be qualified. "a specialization of <code>completion_signatures<ErrorSigs...></code> " also does not really work.	Reword as "if <code>error_types</code> is not a specialization of <code>execution::completion_signatures</code> , or if the template arguments of that specialization contains an element..."	
US 248- 378		33.13.6.4		ed	"Class template" should not be in code font in the title, and "state" should be italicized.	Modify accordingly.	
US 249- 379		33.13.6.4	4	te	It is not clear what bullet 4.6 is – it reads like a requirement on <code>stop_token_type</code> , but if so, this paragraph is a poor place for it.	If it is a requirement, move it under 33.13.6.2.	
US 259- 380		33.13.6.5		ed	Comments in the class definitions should not end in semicolons.	Remove them.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 258- 381		33.13.6.5		te	The wording for task<...>::promise_type::initial_suspend in [task.promise] paragraph 6 (second bullet) may imply that a task is eagerly started, i.e., that the awaiter returned from initial_suspend() immediately starts the scheduling operation and causes the task to be resumed. At the very least the second bullet of the wording should be clarified such that the scheduling operation is only started when the coroutine gets resumed.	Change the specification of task::promise_type::initial_suspend to return std::suspend_always. See LWG4349.	
US 257- 382		33.13.6.5		te	The type task<...>::promise_type has exposition-only members source and token. These can be interpreted as always existing which would be a performance issue for the former and an unnecessary constraints for the latter (because stop tokens aren't required to be default constructible).	Remove the token exposition member entirely and move the source member to the state, making its type optional<stop_source_type> and make it optionally present only if the receiver's stop token type mismatches the stop_source_type::token_type. See LWG4347.	
US 256- 383		33.13.6.5		te	The specification of change_coroutine_scheduler(sched) uses std::exchange to put the scheduler into place (in [task.promise] paragraph 11). The problem is that std::exchange(x, v) expects x to be assignable from v but there is no requirement for scheduler to be assignable.	Change the specification to avoid std::exchange and rather transfer the scheduler using only constructions. See LWG4337.	
US 255- 384		33.13.6.5		te	Normally, the get_allocator query forwards the allocator from the receiver's environment. For task the get_allocator query used for co_awaited senders uses the allocator passed when creating the coroutine or the default if there was none. It should use the receiver's environment, at least, if the receiver's environment supports a get_allocator query. Supporting the receiver's allocator isn't always possible: the used allocator type needs to be known when the coroutine is created. At that time the receiver isn't known, yet. As a result the receiver's environment may provide an allocator which is incompatible with the allocator type used by the coroutine. It may	If the receiver's environment provides a get_allocator query which is compatible with task's scheduler_type use this allocator. See LWG4335.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					be possible to use the receiver's allocator if it is convertible to the allocator type used by the coroutine and to produce a compile-time error otherwise.		
US 254- 385		33.13.6.5		te	Normally the allocator_arg argument has to be the first argument when present. For task<...>::promise_type the allocator_arg can appear at an arbitrary position (except the last because it always needs to be followed by the allocator). This permission is inconsistent and the position of the allocator_arg argument and the allocator should be limited to come first. For containers the optional support for allocators is implemented once for every container. For coroutines the optional support for allocators is implemented once for every coroutine definition. To support an optional allocator the coroutine definition needs to use an allocator and either gets duplicated not using an allocator or a forwarding function is added which adds the default allocator. With the flexible allocator position optional allocator support can be provided using a trailing argument list, i.e., adding , auto&&.... Instead of constraining task it may be more reasonable to add the flexibility to generator.	Constrain the allocator_arg argument to be the first argument. See LWG4334.	
US 253- 386		33.13.6.5		te	Unlike generator the allocator customization of task constraints the allocator type used for the coroutine to be convertible to the configured allocator_type. This prevents easy use of an allocator especially when no allocator is configured and the default (std::allocator<std::byte>) is used. The reason for this constraint is that the get_allocator is forwarded to co_awaited senders and is intended to be the same as the allocator used for the coroutine frame. It may be reasonable to allow use of an arbitrary allocator when there is no explicit configuration of the allocator_type. In this case it may also be	Allow use of any allocator for the coroutine frame from the parameter list, even if this allocator can't be exposed to child operations. See LWG4333.	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					reasonable to not support the get_allocator query when co_awaiting senders.		
US 252- 387		33.13.6.5		te	The function task::promise_type::unhandled_stopped() is called from set_stopped() of a receiver and calls set_stopped itself. These functions are required to be noexcept. Thus, unhandled_stopped() can't throw an exception and should be marked noexcept. All other declarations of unhandled_stopped() are already marked noexcept but task::promise_type::unhandled_stopped() isn't.	Add noexcept to task::promise_type::unhandled_stopped() both in the synopsis and the specification. See LWG4340.	
US 251- 388		33.13.6.5		te	The template parameter V of task::promise_type::return_value doesn't have a default template argument specified. Specifying a default template argument of T would enable use of co_return { ... } which would be consistent with normal return statements. This feature was not discussed in the design paper but based on the LEWG discussion on 2025-08-26 it is considered to be more a bug fix than a new feature.	Add a default argument in the synopsis of return_value: template<typename V = T> void return_value(V&& value); See LWG4345.	
US 250- 389		33.13.6.5		te	The synopsis for std::execution::task<T, E>::promise_type declares return_void() or return_value(V&&). However, there is no specification of what these functions actually do. return_void() doesn't need to do anything at all. return_value(V&& v) needs to store v into the result.	Add specifications for the functions: void return_void(); Effects: does nothing. template<class V> void return_value(V&& v); Effects: Equivalent to result.emplace(std::forward<V>(v)). See LWG4346.	
US 260- 390		33.13.6.5	2	te	The template arguments of the specialization of completion signatures are function types. We want their parameter types.	Add "the parameter types of the" before "template arguments".	
US 261- 391		33.13.6.5	3, 16, 17	te	The parameter's type is const allocator_arg_t&.	Either modify this to refer to the elements of Args, or use the reference type if we want to talk about the parameter.	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02	Document:	Project:
-----------------	-----------	----------

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
FI-392		33.14		te	Scope association concepts were removed from the async scope facility during wording review. Subsequent implementation experience has shown that those concepts are useful and beneficial both for standard library implementations and programmers wishing to write their own async scopes. Bring back the scope association concepts.	Adopt the paper P3815.	
CA-393		33.14.1	Paragraph 2	te	The transformation applied during LEWG review which removed `scope_association` is a potential pessimization.	Adopt P3815, "Add `scope_association` concept to P3149"	
US 262-394		33.15	3	ed	Expressions cannot return things.	Change "returns" to "has the value".	
RO 4-395		33.15: 7,8,10, 11 33.16.02: 4,6		te	The `parallel_scheduler` specification could use a few tweaks to make it better: <ul style="list-style-type: none"> - `receiver_proxy::try_query` could possibly be const-qualified. - `receiver_proxy` does not need a virtual destructor as the object is never destroyed polymorphically - `receiver_proxy::try_query` requires `inplace_stop_token` and doesn't accept an arbitrary stop token - `system_context_replaceability` is not a good name to use for the namespace in which the replaceability APIs lie 	Consider the proposals from P3804R0: Iterating on `parallel_scheduler`: <ul style="list-style-type: none"> • Make try_query const-qualified • Remove virtual destructor from receiver_proxy • Allow other stop tokens to be propagated through try_query • Consider renaming system_context_replaceability Improve working around customizations for bulk	

¹ **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

² **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
					The wording around the customizations of `bulk_unchunked`/`bulk_chunked` for `parallel_scheduler` isn't precise enough.		
US 263- 396		33.16.2	2	te	This should require the shared_ptr to actually own the object. Additionally, "implements the interface" is not C++.	Say "whose type is derived from". Add the ownership requirement.	
US 264- 397		33.16.2	4-7	ed	This should be a separate subclause.	Move these paragraphs into a new subclause "Receiver proxies".	
US 265- 398		33.16.3	1,4,7	te	Storages do not have lifetimes; they have durations.	Change the beginning to "The ends of the lifetimes of *this and the object referred to by r, and the end of the duration of any storage".	
US 266- 399		33.16.3	4,7	ed	The meaning of "one of the expressions below" is not clear (it is meant to exclude execute).	Change to say "the call to set_value, set_error, or set_done on r".	
US 267- 401		Annex C		te	CWG2823 (adopted November 2023) clarified that dereferencing a null or past-the-end pointer is UB, even if the address is immediately taken. CWG2875, which proposes to add an Annex C entry documenting the difference from C, should be resolved since no paper has materialized within the C++26 timeframe to allow this construct in C++.	Apply the proposed resolution for CWG2875.	
US 268- 402		C.1.4	Paragraph 2	te	"Permit the implementation to store backing arrays in static read-only memory."	Change "Permit" to "Require". See comment US 40 for rationale. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2752r3.html	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)



2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2025-10-02

Document:

Project:

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
						<div> <div>  <p>p2752r3.html</p> </div> <div>  <p>p3824r0.html</p> </div> </div>	
AT10 -403		D.15	Paragraphs 4 and 6	te	<code>complex</code> has been made <i>tuple-like</i> (see 22.4.3 and 29.4.9), but the current wording does not support <i>cv-qualified</i> <code>complex</code> .	Add <code><complex></code> to the list of headers that make the <code>volatile-specialization</code> available.	
US 269- 406		Index		ed	If the new notion of "replaceable" is not removed from the Standard, then it should be listed in the Index under "replaceable," which currently lists only the old meaning.	Index, page 2480: Replace: replaceable, 240 with: replaceable, class, 306 function, 240 type, 80	

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Prop
------------------------	----------------	----------------------	----------------------------	---------------------------------	----------	------

File: ISO_IEC CD 14882_AFNOR.docx

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

ISO_IEC CD 14882_AFNOR.docx: Collation successful

ISO_IEC CD 14882_ANSI.docx: Collation successful

ISO_IEC CD 14882_ASI.docx: Collation successful

ISO_IEC CD 14882_ASRO.docx: Collation successful

ISO_IEC CD 14882_BDS.doc: Collation successful

ISO_IEC CD 14882_BSI.doc: Collation successful

ISO_IEC CD 14882_DIN v2.docx: Collation successful

ISO_IEC CD 14882_GOST R V2.doc: Collation successful

ISO_IEC CD 14882_PKN.docx: Collation successful

ISO_IEC CD 14882_SCC.doc: Collation successful

ISO_IEC CD 14882_SFS.docx: Collation successful

ISO_IEC CD 14882_UNE R1.doc: Collation successful

ISO_IEC CD 14882_UNI.doc: Collation successful

ISO_IEC CD 14882_UNMZ.doc: Collation successful

Collation of files was successful. Number of collated files: 14

SELECTED (number of files): 14

PASSED TEST (number of files conformed to CCT table model): 14

FAILED TEST (number of files conformed to CCT table model): 0

ISO_IEC CD 14882_AFNOR.docx: Collation successful

PASSED OTHER FILES (number of files to be collated at the end of the result file not conformed to CCT table model): 1

CCT - Version 2021.1