

Do not remove Contracts from C++26

Timur Doumler (papers@timur.audio)
Joshua Berne (jberne4@bloomberg.net)
Gašper Ažman (gasper.azman@gmail.com)
Peter Bindels (cpp@dascandy.nl)
Peter Dimov (pdimov@gmail.com)
Eric Fiselier (eric@efcs.ca)
Fabio Fracassi (fabio@fracassi.de)
Pablo Halpern (phalpern@qti.qualcomm.com)
Tom Honermann (tom@honermann.net)
John Lakos (jslakos@gmail.com)
Nevin Liber (nliber@anl.gov)
Lisa Lippincott (lisa.e.lippincott@gmail.com)
René Ferdinand Rivera Morell (grafikrobot@gmail.com)
Oliver Rosten (oliver.rosten@gmail.com)
Jan Schultke (janschultke@googlemail.com)
Hui Xie (hui.xie1990@gmail.com)

Document #: P4433R0
Date: 2026-10-23
Project: Programming Language C++
Audience: EWG

On 22 October 2026, the DIS ballot for C++26 concluded. At the time of writing, we do not yet know whether the ballot has met the approval criteria¹ laid out in the ISO/IEC Directives, or has failed to do so, nor do we know the wording of any technical comments submitted by national bodies (NBs). Whatever the case may be, it seems that – for the first time in the history of C++ – some NBs have chosen to vote against the DIS.

It has come to our attention that during the DIS ballot, a small number of individuals have actively lobbied NBs against approving the C++26 DIS due to their own sustained opposition to *contract assertions*, a new language feature added in C++26. These lobbying activities have been accompanied by a public petition [[Falco26](#)] addressing NBs directly, a paper [[P4334R0](#)] that describes "fundamental flaws" in contract assertions, and other papers² published around the same time by the same authors or a subset thereof.

The petition [[Falco26](#)] and the papers accompanying it contain a number of claims, including that C++26 contract assertions lack genuine consensus in WG21; that the design is hasty; that technical objections were not given a proper hearing; and/or that alternative designs were not considered. These claims are not true.

In addition, the description of C++26 contract assertions given in [\[P4334R0\]](#) contains a number of inaccuracies. It misrepresents the specification itself,³ the amount of wording it consists of,⁴ its design intent,⁵ how it relates to hardening⁶ and static analysis,⁷ its existing implementations,⁸ existing practice,⁹ teachability,¹⁰ and proposed extensions.¹¹ None of the listed concerns are new; they have all been previously published by those authors (or a subset thereof), presented to WG21, and extensively discussed there.¹²

The C++26 DIS, including contract assertions, represents the strong, sustained, good-faith consensus of WG21. Contract assertions have been pursued in WG21 since 2004,¹³ and have been developed in their current form since 2019, with an extraordinary amount of effort and diligence dedicated to their design and specification.¹⁴

Normally, WG21 tries to avoid relitigating concerns already heard unless there is new information. On contract assertions, however, WG21 leadership has for the last two years gone to exceptional lengths – far beyond anything the WG21 process requires – to give those presenting these concerns every opportunity to communicate them, at length and repeatedly, to everyone involved. Across multiple WG21 meetings, entire days¹⁵ in the language design subgroup (EWG) have been devoted to making sure these concerns were heard and considered. Even when the repeated objection papers contained no new information and did not present an alternative design¹⁶ that we could have considered instead, significant time was still devoted to seeking a resolution and reconsidering our consensus position.

Despite these repeated rehearsals of the concerns, polls¹⁷ in EWG and in WG21 plenary since February 2025 have consistently been strongly in favour of adopting and retaining contract assertions for C++26. WG21 plenary adopted contract assertions into C++26 with 100 votes in favour and 14 against, and approved the C++26 draft for DIS ballot with 114 votes in favour and 12 against.

The opposition has not merely been "outvoted"; rather, WG21 has carefully considered each individual concern, weighed the trade-offs involved, and ultimately decided, in each case, in favour of the specification that is now in the C++26 DIS. WG21 has repeatedly and clearly expressed that it wants C++26, including contract assertions, in its current form.

It has been suggested in [\[P4238R1\]](#) that returning the C++26 DIS to WG21 over contract assertions would cost no more than "one ballot cycle: a revised draft and a second vote", amounting to "a delay of months". It has further been suggested that contract assertions, once removed, could simply return in C++29 together with their planned extensions. This is not realistic¹⁸.

Removing contract assertions would itself be a major change to the C++26 DIS, requiring drafting and reviewing wording, as well as redesigning the specification for standard-library hardening. More importantly, WG21 has been here before when contracts form C++20 in 2019. It took until 2025 to arrive at a smaller, more refined, and deliberately extensible design that left out major use cases for future extensions. The needed extensions to meet everyone's need depend on having a stable foundation to build upon, and that foundation would be removed by removing Contracts from C++26.

As per the ISO/IEC Directives,¹⁹ the decision on what to do next rests with the SC22 chair, in consultation with the SC22 secretary, the WG21 convenor, and the Office of the CEO.

If the DIS ballot has *passed*, the options are:

- to proceed straight to publication of C++26 as an International Standard (IS),
- to make technical changes and register a revised C++26 draft as a Final Draft International Standard (FDIS),
- to make technical changes and circulate a revised C++26 draft for another (eight-week) DIS ballot.

If the DIS ballot has *failed*, the options are:

- to circulate a revised C++26 draft for another (eight-week) DIS ballot,
- to circulate a revised C++26 Committee Draft (CD) for NB comments,
- to seek approval of C++26 as a Technical Specification (TS) instead of an IS,
- to cancel C++26 entirely.

The role of WG21 in this decision-making process is to offer a recommendation and, if needed, a revised C++26 draft.

This paper is a formal proposal for the consideration of WG21, submitted in time for its upcoming meeting in Búzios (November 2026). WG21 will, as the ISO/IEC Directives require, address every NB comment on the C++26 DIS at that meeting, just as it addressed every NB comment on contract assertions received on the C++26 CD. In this proposal, we urge WG21 not to reopen any language design questions in the C++26 DIS – and the design of contract assertions in particular – in response to comments repeating concerns WG21 has already heard and decided:

- **If the DIS ballot has passed, we should seek to proceed with publication;**
- **If the DIS ballot has failed, we should seek another ballot without altering contract assertions or removing them from C++26;**
- **Any comments asking for the removal of contract assertions from the DIS should be discussed (as procedurally required), but, absent new technical information, WG21 should reject them as having no consensus for change and attach this paper to its response as supporting reasoning.**

We hereby ask WG21 to explicitly confirm the above intention at the Búzios meeting via poll.

NBs are, of course, entitled to review the work of WG21 and to disapprove of it upon review; that is what the DIS ballot is for. However, what seems to have happened here is different: individuals who actively participated in the discussions in WG21, and whose position did not gain consensus there on its technical merits, have lobbied NBs with one-sided representations of the facts in order to obtain through the ballot what they could not obtain in WG21.

None of this is to question anyone's motives. We continue to believe that everyone involved, regardless of their opinion on the C++26 DIS, has the best interests of the C++ community at heart. But good faith on all sides does not change what the ISO/IEC Directives require of

WG21: sustained opposition within the committee must be addressed in good faith, which has happened; it need not be resolved successfully; consensus need not be unanimous; and sustained opposition does not confer a veto.²⁰

The question now before WG21 is therefore not whether NBs may vote as they did, but what WG21 should recommend in response. We see no reason for that recommendation to differ from the position WG21 has reaffirmed at every opportunity since February 2025.

If the C++ programming language is to have a future, its contributors must be able to trust that open, thorough technical work carried by a strong consensus in WG21 is allowed to successfully complete and cannot be undone at the DIS ballot by a small group of individuals unhappy with the results of discussions they were given every opportunity to influence within WG21.

We encourage WG21 to reaffirm the consensus established through years of rigorous technical review, ensuring stability and predictability for implementers, tool vendors, and the broader C++ community. C++26 needs to ship, with its intended design intact. We urge WG21 to do its part to make that happen.

Notes

1. See ISO/IEC Directives, Part 1, section [2.6.3](#):

An enquiry draft is approved if

- a) a two-thirds majority of the votes cast by the P-members are in favour, and
- b) not more than one-quarter of the total number of votes cast are negative.

Abstentions are excluded when the votes are counted, as well as negative votes not accompanied by any comments of a technical nature.

2. This includes [\[P4381R0\]](#), [\[P4330R0\]](#), and [\[P4238R1\]](#).

3. [\[P4334R0\]](#) claims that C++26 contract assertions "includes a completely untried inheritance model". C++26 contract assertions deliberately *do not* include a model for contract inheritance (untried or otherwise). The model proposed in [\[P3097R3\]](#) – an evolution of the Eiffel model from the 1980s – was initially approved for C++26 by EWG with strong consensus, but later removed and postponed until C++29 to address the sustained objections by some of the authors of [\[P4334R0\]](#) (see [\[P3573R0\]](#) and [\[P3616R0\]](#)).

[\[P4334R0\]](#) claims that C++26 contract assertions "leads to new forms of UB". [\[P4334R0\]](#) itself, and the papers it cites, do not seem to provide any concrete examples of newly introduced UB. Introducing no new forms of UB is a core design principle of C++ contract assertions (see [\[P2900R14\]](#) Principle 13, in Section 3.1.4).

What contract assertions *do* allow is the use of expressions written in the existing C++ language as assertion predicates – and C++ expressions can have UB when evaluated. Preventing this would require a dialect of C++ that is completely free of UB while still being expressive enough for everyday code. Attempts to achieve this daunting goal ([\[P2680R1\]](#), [\[P3390R0\]](#)) have so far not been successful, despite significant time and effort spent. That goal is still being actively pursued ([\[P3874R1\]](#)); if it ever succeeds, it will be a useful piece of functionality completely orthogonal to contract assertions.

[\[P4334R0\]](#) claims that C++26 contract assertions "offers new ways of making errors through inconsistent application in TUs". As discussed in more detail in [\[P3846R1\]](#) Concerns 2 and 4, the default behaviour for functions defined in multiple TUs is neither a "new way of making an error" nor specific to contract assertions, but merely a consequence of the existing C++ linkage model. The specification in the C++26 DIS gives implementations all the necessary leeway to improve on this situation, thereby providing a significant net improvement over pre-C++26 (macro-based) assertions. It would, however, be inappropriate to normatively prescribe a more deterministic linkage model for C++ as part of the specification for contract assertions.

4. [P4334R0] claims that C++26 contracts are "bloated" because they add "56 pages" of wording to the Standard. While it is true that the wording section of [P2900R14] consists of that number of pages, that is only because it includes extensive amounts of pre-existing wording reproduced for context around the areas of insertion/deletion. If we count the net number of words actually added to the C++ working draft by recent major features as standardised in their initial version, it becomes apparent that contract assertions are a comparatively lightweight feature:

C++20 Coroutines – 3,452 words
C++26 Contract assertions – 4,946 words
C++20 Modules – 8,043 words
C++20 Concepts – 13,023 words
C++26 Reflection – 18,966 words
C++26 Execution – 29,427 words
C++20 Ranges – 41,863 words

One reason that the wording for C++26 contract assertions is not even more compact is that it also includes a number of cleanups and refactorings in existing parts of the C++ language (function return values, coroutines, etc.).

5. [P4334R0] advocates for providing whole-TU or whole-program *guarantees* based on a combination of static analysis and enforced range checks as a modern path towards software safety. Indeed, for a memory-safe programming language, such guarantees are necessary (but not sufficient; see [P3874R1]). However, [P4334R0] goes on to criticise C++26 contract assertions because they do not provide such guarantees. Doing so is a logical fallacy, akin to criticising static reflection for not providing runtime exception handling.

Assertions, as a programming technique, are not a "threat" to approaches that provide guarantees; rather, they are designed to *complement* such approaches. They are designed to let the user augment their program with optional, configurable runtime correctness checks in cases where non-optional checks, static proofs, and safety constraints expressed via the type system are not practical. In cases where those techniques *are* practical, they are often the best approach, and assertions take nothing away from their utility. Assertions in general – and C++26 contract assertions in particular – thus *add* a layer of safety, primarily targeting *functional safety* as opposed to language-level safety (see [P3500R1], [P3578R1], [P3846R1] Concern 1).

6. C++26 standard-library hardening [P3471R4] is specified in terms of C++26 contract assertions and that specification had strong consensus in WG21. Nevertheless, [P4334R0] states that C++26 contracts "isn't integrated with or appropriate for hardened libraries" and cites [P3878R0] and [Dionne25] in support of that claim.

The cited paper [P3878R0] has been superseded by its revision [P3878R1], which removes the criticism of contract assertions as a feature while retaining the same normative wording fix: an implementation of the C++ standard library should not be considered *hardened* if it evaluates hardened preconditions with the *observe* contract-evaluation semantic, but only if

it does so with the *enforce* or *quick-enforce* semantic. That fix (which does not change in any way the specification for C++26 contract assertions themselves) was applied to C++26 in Kona (November 2025) as part of the resolution of CD ballot comments, before C++26 went out for DIS ballot. Nowhere does [\[P3878R1\]](#) state that C++26 contract assertions are not appropriate as a basis for specifying standard-library hardening.

[\[Dionne25\]](#) is an article co-authored by the developers of libcpp – who have contributed significantly to the design of C++26 contract assertions – and talks about "hardening the C++ Standard Library at massive scale". Here is what that paper actually says:

Notably, the Standard leverages another C++26 feature, Contracts, which provides an extensive framework for specifying program invariants and handling violations. That gives developers significant flexibility in how they handle a failing hardening check. Contracts were designed with consideration for Library Hardening as a use case, ensuring that libcpp assertion failures can be modeled directly by the Contracts evaluation semantics (specifically, the trapping mechanism used in libcpp hardening is precisely the quick-enforce evaluation semantic).

7. [\[P4334R0\]](#) claims that C++26 contract assertions "hasn't been used to support static analysis". In fact, they have been developed in direct collaboration with static analysis experts (see [\[P3386R1\]](#)) and used by state-of-the-art static analysis tools (see [\[Martin25\]](#)).

8. [\[P4334R0\]](#) calls C++26 contract assertions "unimplemented", which is not true. C++26 contract assertions have complete implementations in two major compilers, GCC and Clang.

The GCC implementation was released in April 2026 as part of GCC 16.1, and shipped as the system compiler on Ubuntu 26.04. [\[P4334R0\]](#) claims that GCC 16.1 "ships it only as experimental". This critique is misleading: GCC labels its support for *every* C++26 feature as experimental – including e.g. reflection ("Because the ISO C++26 standard is very recent, GCC's support is experimental", see <https://gcc.gnu.org/projects/cxx-status.html>), and it labelled its C++20 support as experimental until GCC 16 (see <https://gcc.gnu.org/gcc-16/changes.html>), more than five years after C++20 was published.

The Clang implementation is still being upstreamed and is not yet part of a Clang release, but it has been publicly available on [GitHub](#) and on [Compiler Explorer](#) since October 2024.

[\[P4334R0\]](#) further states that "Microsoft and EDG have objected to standardizing" C++26 contract assertions. The sustained objection by Microsoft is well-known ([\[P3506R0\]](#)). EDG have since open-sourced their C++ front end, and a branch of the EDG front end with an implementation of C++26 contract assertions, maintained by one of the feature's authors, is likewise already available on [GitHub](#) and [Compiler Explorer](#).

[\[P4334R0\]](#) claims that "every feature must have a reasonably obvious implementation" and that C++26 contract assertions violate that rule. However, the paper fails to point out which parts of the feature it considers non-obvious to implement and how the feature fares worse

on this axis than other features we have successfully standardised in the past. The actual implementers have reported no significant hurdles; on the contrary, they report that contract assertions are a straightforward feature to implement. The GCC and Clang implementers' report [P3460R0] states that "P2900's specification was clear and implementable" and that "constification was trivial and unsurprising to implement in both GCC and Clang".

Further, there are well-known examples of C++ features, such as `constexpr`, that *were* difficult to implement (certainly much more so than contract assertions) and became a huge success regardless.

Finally, [P4334R0] claims that a C++ feature should "always provide a transition path" and that C++26 contract assertions violate that rule. In fact, C++26 contract assertions have been explicitly designed for gradual adoption in legacy codebases of any coding style and any quality (see [P2899R1], Section 2.8).

9. [P4334R0] claims that C++26 contract assertions are "untried". The same claim was previously made in [P3849R0] and [P3851R0], and before that in [P3173R0], [P3506R0], and [P3573R0]. As discussed multiple times in EWG, this claim is incorrect; a detailed response is available in [P3846R1] Concern 17.

C++26 contract assertions were tried on multiple large codebases ([P3460R0], [P3336R0]) and found several pre-existing bugs in those codebases. Deployment experience was gathered to the extent possible for a core language feature that includes new declaration syntax but is not yet part of a published ISO standard. Decades of experience exist with a *subset* of that feature (e.g., `BSLS_ASSERT`).

Rather than asking whether C++26 contract assertions are "untried", it seems more helpful to ask whether they are *less* tried at this stage in their standardisation than other language features we have successfully added to C++ (lambda expressions, `constexpr`, reflection...).

10. [P4334R0] claims that C++26 contract assertions "hasn't been demonstrated to be easily teachable". This claim ignores the teaching materials that are already available, such as several well-received CppCon talks ([Lippincott24], [Sutter25]) and articles ([Doumler25], [Pusz26]).

11. [P4334R0] repeats similar claims from [P3829R0], [P3835R0], and [P3851R0] that C++26 contract assertions are "incomplete". These claims were discussed multiple times in EWG; a detailed response is available in [P3846R1] Concern 14.

The "incompleteness" is deliberate and should be understood as intentionally designing for extensibility. Shipping all sub-features that anyone might want in the first release is neither realistic nor how WG21 has ever operated. Every other major feature shipping in C++26 is "incomplete" in the same sense. To name just one missing piece of each: `std::simd` does not yet support user-defined element types, execution does not yet provide asynchronous

I/O and timers, and reflection does not yet include code generation. As with contract assertions, the missing parts will be added in C++29 and beyond. This incremental standardisation has been successful in the past: for example, C++20 coroutines shipped without library support, C++11 constexpr shipped in an extremely limited form.

The claim in [P4334R0] that the C++29 extensions planned for contract assertions (see [P3850R1] for an EWG-approved roadmap) are "ad-hoc patches" is factually incorrect: all of those extensions are purely additive and fully backwards compatible, and C++26 contract assertions have been designed from the start with those extensions in mind (see [P2755R1]). In addition, several of the papers that [P4334R0] cites as proposals to modify C++26 contract assertions do not, in fact, propose any changes to C++.

The claim in [P4334R0] that "the pieces ordinary use needs are not in the feature but deferred to C++29-targeted extensions still being designed" is also incorrect. The paper cites three extensions as examples of this: contract assertions on virtual functions [P3097R3], postcondition captures [P3098R2], and implicit contract assertions [P3100R9]. The first of these extensions was adopted into the C++29 working draft in Brno (June 2026), two months before [P4334R0] was published; the second has been design-complete for years; the third targets an entirely different class of bugs than pre, post, and contract_assert do, and is in no way required in order to use them.

Finally, the claim in [P4334R0] that contract assertions are "endlessly extended and changed" is at odds with the fact that [P2900R14], adopted in February 2025, remains the final revision of the proposal. The *only* net change to C++26 contract assertions since its adoption has been the removal of a single function, `evaluation_exception()`, from the contract-violation handling API in response to CD ballot comments [P3819R0], adopted in Kona (November 2025) as a pure standard-library fix with no effect on the wording of the core language feature.

12. See [P3846R1] for references to all previous "contracts concerns" papers and NB comments by the authors of [Falco26] and [P4334R0].

13. The first formal WG21 proposal to add contract assertions to C++ was [N1613], published in 2004.

14. The extensive history and evolution of C++ contract assertions throughout their 22 years of development is chronicled in detail in [P2899R1]. That paper also records the polls taken on each and every individual design question along the way: in SG21 at more than 40 teleconferences and at in-person sessions at WG21 meetings from Prague (February 2020) to Wrocław (November 2024), and in EWG at every WG21 meeting from Tokyo (March 2024) to Hagenberg (February 2025), where contract assertions were adopted into C++26.

The contract-assertion proposal itself [P2900R14] contains a complete revision history between September 2023 and February 2025 when it was adopted. Its direct precursor papers ([P2388R4], [P2461R1], [P2521R5]) contain the revision history from the point when

the post-C++20 design began to take shape until it was consolidated in P2900. The available design options for addressing the concerns now repeated in [\[P4334R0\]](#) have individually been thoroughly investigated and the results of those investigations published in their own papers, for example [\[P3261R2\]](#) for const-ification.

Together these papers evidence not a hasty or "half-baked" feature, but many years of extremely thorough language design work that extensively covered the available design space.

15. Papers dedicated specifically to concerns over C++26 contract assertions, including those reiterated in [\[P4334R0\]](#), were presented and discussed in EWG at the following WG21 meetings:

- Wednesday in Tokyo (March 2024)
- Tuesday in St. Louis (June 2024)
- Monday afternoon and Tuesday in Wrocław (November 2024)
- Tuesday in Hagenberg (February 2025)
- Monday and Tuesday in Kona (November 2025)
- Monday afternoon in Croydon (March 2026)

At each of these meetings, as is the standard practice in WG21, all concerns were given ample time for discussion and no poll was taken until the chairs believed the room had heard and understood what had been presented. The polls taken at each of these meetings, and their outcomes, are listed in note 17 below.

16. [\[P4334R0\]](#) refers multiple times to [\[P4324R0\]](#) as a preferable alternative design to C++26 contract assertions. That paper is a design sketch with no wording, describing itself as an "exploration". The first revision of that paper was published in August 2026 – 18 months after the C++26 feature-freeze, 14 months after WG21 approved C++26 for CD ballot, and five months after WG21 approved C++26 for DIS ballot. It has not yet been reviewed by any subgroup in WG21.

Moreover, [\[P4324R0\]](#) is not an alternative to C++26 contract assertions; it builds on them. It retains the C++26 pre, post, and contract_assert syntax with an added template argument ("the syntactic keying used here is similar to P3400"), reuses the C++26 evaluation semantics, assertion kinds, detection modes, and contract-violation handler, and its implementation is a fork of GCC that modifies GCC's existing C++26 contracts implementation rather than replacing it. What the paper explores is not a fully library-based, alternative contracts facility, but rather a different mechanism for controlling the properties of C++26 contract assertions from library code, in the same design space as the extension proposal [\[P3400R4\]](#) (also known as "labels") currently under design review in EWG for inclusion in C++29.

17. The following polls on C++26 contract assertions in EWG and in WG21 plenary are recorded in the WG21 meeting minutes ([\[N5007\]](#), [\[N5031\]](#), [\[N5040\]](#)) and in the WG21 paper

tracker ([Issue 1648](#) and related issues). Results are given as SF/F/N/A/SA (strongly in favour / in favour / neutral / against / strongly against).

- **Tokyo (March 2024):** EWG, non-binding feedback polls after a full day of design review, including: "contracts should have enforce semantics only, and should not have ignore nor observe": 6/1/3/15/24; "contracts should expose less undefined behavior than regular C++ code does": 17/12/12/12/5; "there should be some usage experience of contracts in an implementation of the STL [...] before contracts can move to plenary": 16/30/12/3/0 – a request that was subsequently met by [\[P3460R0\]](#).
- **St. Louis (June 2024):** EWG, after a day of discussion, took polls on proposed design changes (restricting the number of evaluations, specifying termination behaviour, disallowing exceptions from violation handlers, removing const-ification, reserving an assume semantic): none gained consensus; for example, "contracts should not be able to evaluate preconditions/postconditions/assertions more than once per invocation": 3/2/2/17/11, consensus against. For another half a day at the same meeting, SG21 and EWG both dived deep into the discussion of virtual functions, and in the end approved the design in P3097R0, 23/11/3/5/2, consensus.
- **Wrocław (November 2024):** EWG devoted another day and a half to contracts: "the contracts proposal in P2900's Minimal Viable Product shall be changed to incorporate stricter contracts in addition to regular contracts": 10/6/3/14/16, consensus against; "P2900r11: send to CWG and LEWG for inclusion in C++26": 25/17/0/3/12, consensus.
- **Hagenberg (February 2025):** EWG, after discussing [\[P3573R0\]](#), [\[P3506R0\]](#), and further concerns papers: "remove constification": 9/7/6/37/14, consensus against; "reduce the amount of Undefined Behavior in contracts": 9/7/9/22/19, consensus against; "add ODR to contracts": 6/5/10/25/17, consensus against; "disallow pre/post contracts on virtual functions entirely": 20/24/13/14/2, consensus in favour (see note 3); "remove P2900 from CWG's consideration for C++26, find a different ship vehicle": 9/8/3/19/41, consensus against. WG21 plenary then adopted [\[P2900R14\]](#) into C++26 with 100 votes in favour, 14 against, and 12 abstentions.
- **Kona (November 2025):** EWG, after processing the 23 NB comments on contract assertions received in the CD ballot (library-related comments were processed separately by LEWG; see notes 6 and 11): "Sweden SE 6.11 [basic.contract] Concerns about contracts — we want to pursue a change for C++26": 12/3/1/15/26, no consensus for change. At the same meeting, WG21 removed a different feature, trivial relocation, from C++26 in response to NB comments, evidencing that the process *does* remove features from the working draft when there is consensus to do so.
- **Croydon (March 2026):** EWG: "Instruct CWG to Remove the Contracts Facility (as added by P2900R14 and subsequent proposals) from C++26, as a response to RO 2-056 and reject NL 6.11": 11/3/7/24/35, not consensus. WG21 plenary then approved the C++26 draft for DIS ballot with 114 votes in favour, 12 against, and 3 abstentions.

18. At the time of writing, there is no wording for removing contract assertions from C++26, and producing it is not a purely mechanical exercise. It would require:

- removing the core-language wording for C++26 contract assertions;
- either reverting or re-reviewing the cleanups and refactorings of pre-existing wording in areas such as function return values, replaceable functions, and coroutines that [\[P2900R14\]](#) introduced;
- removing contract assertions on virtual functions [\[P3097R3\]](#) from the C++29 working draft, into which they were adopted in Brno (June 2026);
- removing or respecifying C++26 standard-library hardening [\[P3471R4\]](#), which is written in terms of contract assertions and has never had complete, reviewed wording that is independent of contract assertions.

All of this would have to be drafted, reviewed by EWG, LEWG, CWG, and LWG, and approved by WG21 plenary prior to any ability to submit a new draft. That draft, without contract assertions, would then have to be approved in another round of NB ballot.

WG21 has been through this cycle before. In 2019, it removed contracts from the C++20 working draft [\[P1823R0\]](#) because, in the words of that paper, "even the initial authors of Contracts for C++20 have no agreement" and there was "no implementation experience"; its authors noted that WG21 had "removed features before (such as Concepts for C++11) and came back with better approaches". Concepts, removed from C++0x in 2009, returned in C++20. For contract assertions, coming back took more than five years: SG21 was formed, use cases were collected, and a deliberately minimal and extensible design was developed and reviewed. The C++26 design is smaller than the C++20 one – it has no axioms, no build levels, and no contracts on virtual functions – but it is far more carefully specified, has implementations in two major compilers, and was adopted with 100 votes in favour and 14 against. None of the reasons for the 2019 removal applies today, and nothing suggests that a second removal would be followed by a faster return or successful delivery of a more complete feature if there even is any successor attempt.

19. See ISO/IEC Directives, Part 1, section [2.6.4](#):

On receipt of the results of the voting and any comments, an appropriate group of the committee, which would typically be the working group to which the project is assigned, will, at a meeting, address any comments received and prepare, as needed, a revised version. The group will advise the Chair on the possible courses of action. The Chair of the committee, in consultation with the Secretary/Committee Manager and the Project Leader or working group Convenor, and the Office of the CEO, shall take one of the following courses of action:

- a) when the approval criteria of 2.6.3 are met, and no technical changes are to be included, to proceed directly to publication (see 2.8);
- b) when the approval criteria of 2.6.3 are met, but technical changes are to be included:
 - 1) to register the revised enquiry draft, as modified, as a final draft International Standard, or
 - 2) to circulate the revised enquiry draft for voting (see 2.6.1). The revised enquiry draft will be circulated for a voting period of eight weeks. Committees are limited to only one revised enquiry draft where the approval criteria are met in this case;

- c) when the approval criteria of 2.6.3 are not met:
 - 1) to circulate the revised enquiry draft for eight weeks voting (see 2.6.1), or
 - 2) to circulate the revised committee draft for comments, or
 - 3) to circulate a revised draft as a draft Technical Specification, or
 - 4) to cancel the project, subject to a decision by the committee.

20. See ISO/IEC Directives, Part 1, section [2.5.7](#):

The concept of sustained opposition is not applicable in the context of National Body votes on enquiry or approval ballots since these are subject to the applicable voting rules. [...] If the leadership determines that there is a sustained opposition, it shall try and resolve it in good faith. However, a sustained opposition is not akin to a right to veto. The obligation to address the sustained oppositions does not imply an obligation to resolve them successfully.

Bibliography

- [[Dionne25](#)] Louis Dionne, Alex Rebert, Max Shavrick, and Konstantin Varlamov: "Practical Security in Production. Hardening the C++ Standard Library at massive scale". 2025-11-07
- [[Doumle25](#)] Timur Doumle: "Contracts for C++ explained in 5 minutes". 2025-01-30
- [[Falco26](#)] Vinnie Falco, John Spicer, J. Daniel Garcia Sanchez, and Ville Voutilainen: "C++26 Needs To Go Back". 2026-08
- [[Lippincott24](#)] Lisa Lippincott: "Perspectives on Contracts for C++". CppCon 2024
- [[Martin25](#)] Peter Martin and Mike Fairhurst: "Catching Bugs Early: Validating C++ Contracts with Static Analysis". CppCon 2025
- [[N1613](#)] Thorsten Ottosen: "Proposal to add Design by Contract to C++". 2004-03-29
- [[N5007](#)] Nina Dinka Ranns: "WG21 February 2025 Hybrid meeting Minutes of Meeting". 2025-03-03
- [[N5031](#)] Nina Dinka Ranns: "WG21 November 2025 Kona Hybrid meeting Minutes of Meeting". 2025-11-28
- [[N5040](#)] Braden Ganetsky: "WG21 2026-03 Croydon Hybrid Meeting Minutes". 2026-04-07
- [[P1823R0](#)] Nicolai Josuttis, Ville Voutilainen, Roger Orr, Daveed Vandevoorde, John Spicer, and Christopher Di Bella: "Remove Contracts from C++20". 2019-07-18
- [[P2388R4](#)] Andrzej Krzemiński and Gašper Ažman: "Minimum Contract Support: either *No_eval* or *Eval_and_abort*". 2021-11-15d
- [[P2461R1](#)] Gašper Ažman, Caleb Sunstrum, and Bronek Kozicki: "Closure-Based Syntax for Contracts". 2021-11-15
- [[P2521R5](#)] Andrzej Krzemiński, Gašper Ažman, Joshua Berne, Bronek Kozicki, Ryan McDougall, and Caleb Sunstrum: "Contract support — Record of SG21 consensus". 2023-08-15
- [[P2680R1](#)] Gabriel Dos Reis: "Contracts for C++: Prioritizing Safety". 2022-12-15
- [[P2755R1](#)] Joshua Berne, Jake Fevold, and John Lakos: "A Bold Plan for a Complete Contracts Facility". 2024-04-11
- [[P2899R1](#)] Joshua Berne, Timur Doumle, Rostislav Khlebnikov, and Andrzej Krzemiński: "Contracts for C++ — Rationale". 2025-03-14
- [[P2900R14](#)] Joshua Berne, Timur Doumle, Andrzej Krzemiński, et al.: "Contracts for C++". 2025-02-13
- [[P3097R3](#)] Timur Doumle, Joshua Berne, and Gašper Ažman: "Contracts for C++: Virtual functions". 2026-06-12
- [[P3098R2](#)] Timur Doumle, Gašper Ažman, and Joshua Berne: "Contracts for C++: Postcondition Captures". 2026-05-12
- [[P3100R9](#)] Timur Doumle and Joshua Berne: "A framework for systematically addressing undefined behaviour in the C++ Standard". 2026-10-23
- [[P3173R0](#)] Gabriel Dos Reis: "P2900R6 May Be Minimal, but It Is Not Viable". 2024-02-15
- [[P3261R2](#)] Joshua Berne: "Revisiting const-ification in Contract Assertions". 2024-11-25

[[P3336R0](#)] Joshua Berne: "Usage Experience for Contracts with BDE". 2024-06-23

[[P3386R1](#)] Joshua Berne: "Static Analysis of Contracts with P2900". 2024-11-25

[[P3390R0](#)] Sean Baxter and Christian Mazakas: "Safe C++". 2024-09-11

[[P3400R4](#)] Joshua Berne: "Controlling Contract-Assertion Properties". 2026-07-15

[[P3460R0](#)] Eric Fiselier, Nina Dinka Ranns, and Iain Sandoe: "C++ Contracts — Implementers Report". 2024-10-16

[[P3471R4](#)] Konstantin Varlamov and Louis Dionne: "Standard library hardening". 2025-02-14

[[P3500R1](#)] Timur Doumler, Gašper Ažman, Joshua Berne, and Ryan McDougall: "Are Contracts 'safe'?". 2025-02-09

[[P3506R0](#)] Gabriel Dos Reis: "P2900 Is Still not Ready for C++26". 2024-11-19

[[P3573R0](#)] Michael Hava, J. Daniel Garcia Sanchez, Ran Regev, Gabriel Dos Reis, John Spicer, Bjarne Stroustrup, J.C. van Winkel, David Vandevoorde, and Ville Voutilainen: "Contract concerns". 2025-01-12

[[P3578R1](#)] Ryan McDougall: "What is Safety?". 2025-10-01

[[P3616R0](#)] Ville Voutilainen: "Remove contracts on virtual functions for now". 2025-02-08

[[P3819R0](#)] Peter Bindels, Timur Doumler, Joshua Berne, Eric Fiselier, and Iain Sandoe: "Remove `evaluation_exception()` from contract-violation handling for C++26". 2025-09-02

[[P3829R0](#)] David Chisnall, John Spicer, Ville Voutilainen, Gabriel Dos Reis, and Jose Daniel Garcia Sanchez: "Contracts do not belong in the language". 2025-09-02

[[P3835R0](#)] John Spicer, Ville Voutilainen, and Jose Daniel Garcia Sanchez: "Contracts make C++ less safe — full stop!". 2025-09-03

[[P3846R1](#)] Timur Doumler, Joshua Berne, et al.: "C++26 Contract Assertions, Reasserted". 2025-11-03

[[P3849R0](#)] Harald Achitz: "SIS/TK611 considerations on Contract Assertions". 2025-09-27

[[P3850R1](#)] Timur Doumler and Joshua Berne: "A proposed plan for extending Contracts in C++29". 2026-07-15

[[P3851R0](#)] J. Daniel Garcia, Jose Gomez, Raul Huertas, Javier Lopez-Gomez, Jesus Martinez, Francisco Palomo, and Victor Sanchez: "Position on contracts assertion for C++26". 2025-09-29

[[P3874R1](#)] Jon Bauman, Timur Doumler, Nevin Liber, Ryan McDougall, Pablo Halpern, Jeff Garland, and Jonathan Müller: "Should C++ be a memory-safe language?". 2026-02-23

[[P3878R0](#)] Ville Voutilainen, Jonathan Wakely, John Spicer, and Stephan T. Lavavej: "C++26 Contracts are not a good fit for standard library hardening". 2025-10-07

[[P3878R1](#)] Ville Voutilainen, Jonathan Wakely, John Spicer, and Stephan T. Lavavej: "Standard library hardening should not use the 'observe' semantic". 2025-11-06

[[P4238R1](#)] Vinnie Falco, Ville Voutilainen, J. Daniel Garcia Sanchez, and John Spicer: "Returning C++26 for the Evaluation It Skipped". 2026-09-01

[[P4324R0](#)] Ville Voutilainen: "Exploration of a more library-oriented approach to contracts". 2026-08-14

[[P4330R0](#)] Vinnie Falco: "Analysis of Contracts Papers". 2026-07-28

[[P4334R0](#)] Bjarne Stroustrup, J. Daniel Garcia Sanchez, Vinnie Falco, John Spicer, and Ville Voutilainen: "P2900 Contracts' fundamental flaws". 2026-08-09

[[P4381R0](#)] John Spicer: "Chairing SG21 and then objecting to Contracts". 2026-09-19

[[Pusz26](#)] Mateusz Pusz: "How C++26 contracts work". 2026-09-23

[[Sutter25](#)] Herb Sutter: "The Joy of C++26 Contracts (and Some Myth-Conceptions)".
2025-11