Document Number:	P2659R1
Date:	2022-10-27
Revises:	P2659R0
Audience:	EWG, SG21 (Contracts)
Reply to:	Brian Bi <bbi10@bloomberg.net>
	Alisdair Meredith <ameredith1@bloomberg.net>

A Proposal to Publish a Technical Specification for Contracts

Abstract

We propose that the Committee should publish a Technical Specification on Extensions to C++ for Contracts based on three sources of wording.

1. [P0542R5] and [P1323R2], which were incorporated into the C++20 Working Draft before being removed by [P1823R0], plus minor changes introduced by the project editor
2. [P1607R1], which was approved by EWG and was subsequently in the process of being reviewed by CWG when Contracts was pulled from C++20
3. [P1344R1], which was approved by CWG but not yet applied at the time when Contracts was pulled from C++20

To support a lightweight process that does not distract from ongoing SG21 work, the design for the proposed TS relies *exclusively* on a feature set previously approved by EWG and reviewed by Core.

Revision History

R1

- Added an overview section
- Added more detailed background as a separate section

- Expanded and clarified the motivation
- Added a section stating what questions we expect the TS to answer
- Added a section delineating the schedule on which we hope the TS will proceed
- Added a conclusion
- Added an appendix addressing the questions posed by P2000R3

Overview of Our Objectives

Our overarching goal is to enable the Committee to ship a useful runtime contract checking (a.k.a. Contracts) feature in C++26. We argue that the Committee can best achieve this goal by quickly publishing in parallel a TS based on a design previously approved by EWG and consisting mostly of CWG-approved wording. We deliberately chose this design (1) to facilitate Committee votes on this TS with minimal delay and thus to maximize the information-gathering potential of the TS and (2) to minimize the time required of SG21, EWG, and CWG for technical debate and to avoid interfering with their ability to deliver the full feature in C++26. *This proposal is intended to complement and inform SG21's work, not to compete with or bypass it.* Contracts as a whole, including features shared with the MVP (e.g., varying build modes) and potential future evolution (e.g., violation-handler configuration) will gain valuable real-world experience, whereas they have hitherto been left largely unexplored.

Background

The attempt to add support for Contracts to C++ has a long history, yet that collective effort has failed to make a specification available to the wider community.

1. The first proposal for C++11, [N1773], did not make the feature-complete cut-off, and work ceased.
2. The second major attempt, [N4075], comprised improved precondition checking with a better `assert` facility, targeted for Library Fundamentals TS v2. This effort died in plenary because the Committee did not reach consensus to add a new macro-based facility.

3. The third major attempt, [P0542R5], was incorporated into the C++20 Working Draft after the Rapperswil meeting in June 2018; however, the point was made in [P1823R0] that EWG was still approving major design changes as late as July 2019, and that the active work at the very end of the release cycle made it unclear whether the changes were ready for release. Consensus was reached to remove Contracts from C++20.

With the formation of SG21, a fourth major attempt to produce a working proposal for Contracts was born. SG21 is struggling to make progress in an area with many competing interests and, although consensus has been reached on several points (e.g., [P2521R2]), a full initial proposal has not materialized after an entire three-year Standard development cycle.

Motivation

To obtain feedback based on real-world experience, we propose turning the work that was approved for C++20 into a TS, which could then inform SG21's work toward a concrete proposal for C++26. This approach would be the same as that used for three other successful efforts: Modules, Concepts, and Coroutines. In some cases, the TS feedback led to substantial modifications, resulting in those features being successfully adopted late in the Standard cycle with a high degree of confidence.

Publishing a Contracts TS would allow users to begin using a contract-checking facility, thus providing much wider real-world experience than has been possible to date. Such experience will likely result in consensus about what features belong to a first incarnation of the C++ contracts facility and maximize its usefulness to a wide range of C++ programmers. To date, any such consensus has been manifestly difficult to achieve, and the lack of real-world experience has been a stumbling block to the resolution of perennial disputes over the syntax and semantics of Contracts.

When Contracts was pulled from C++20, a full GCC implementation of Contracts, supporting the behavior that had been approved by EWG, had already been produced (see [P1680R0]) and was in final review for inclusion in GCC 10.0; however, because the Committee has been unable to reach consensus to publish a specification, GCC will not release a feature that would become its own proprietary language extension. Other compilers, such as Clang, have a similar approach toward merging features that lack an official specification. Thus, the removal of Contracts has prevented that

implementation from becoming available to practicing C++ programmers. Since our proposal is deliberately close to the work that was included in (but later removed from) C++20, we expect that a full GCC implementation can be completed and reviewed by the time the proposed Contracts TS is published and would, therefore, be ready for release soon after publication.

Due to the slow progress thus far in SG21, publishing a TS would be of enormous value in advancing the state of Contracts so that a useful product can be delivered by C++26. Although the proposed TS and its implementation provide strong support for runtime validation of preconditions and assertions, their support for postconditions is known to be inadequate for a complete proposal that would fully support static analysis and other tooling. Gaining real-world experience will provide a better understanding of the shortcomings of this specific model, which will in turn allow users to better inform SG21 about any unpredicted issues that were experienced and about predicted issues that turned out to be illusory.

The motivation for a Contracts TS is very similar to the motivation for the Coroutines TS, in which a working specification existed but some members of the Committee wanted to explore a different direction. Publishing a TS will allow progress in the direction already approved by EWG, while work on the second direction can continue in parallel.

The initial draft of the proposed Contracts TS has been published as a separate paper, [P2660R0]. An explicit goal was for this TS to conform as closely as possible to the aforementioned sources, with changes only where strictly necessary to produce a coherent whole that is based on the C++23 DIS. This way, the Committee can guarantee an objective starting point that consists of only those features that were already approved by EWG.[1] We hope that this clean, objective approach will facilitate having the TS ready for balloting at the meeting after C++23 is published, thereby giving the TS the longest possible lead time for C++26.

We have also produced a third paper, [P2661R0], which proposes amendments to the TS that capture the evolution that has successfully gained consensus in SG21. Note that the proposed amendments aim to capture the improvements in understanding that have stemmed from continued discussion but do not attempt to reduce the scope of the TS to a minimal product.

[1] One notable exception is that the `inform` contract behavior was renamed to `observe` so that each of the four contract behavior names is a verb that now has the contract as its direct object, reflecting an informal consensus that was not brought to a vote in EWG because of the removal of Contracts.

Questions for a TS to Answer

We expect that publishing this TS will inform future work on a variety of questions, particularly those listed below, using real-world implementation experience that cannot be obtained without a TS.

- How will implementations support a language feature that requires users to be provided with the ability to supply compiler switches? What changes are needed to the specification to maximize the implementability and usefulness of such features?
- What ABI-compatibility challenges will Contracts pose, and how well will implementations handle them? In particular, is there a viable implementation strategy that enables introducing the use of Contracts to an ABI-frozen library, and how will implementations evolve to support mixed-mode executables?
- To what extent will contract checks, expressed as attributes, be mistakenly perceived as possibly being ignored by a conforming compiler?
- Contract conditions are not part of function types and cannot be applied to function pointers. To what extent is the anticipated usefulness of Contracts limited by these restrictions?
- What are the security implications of failed contract checks and particularly of executing a user-supplied violation handler? How can we mitigate any perceived security risks that might be introduced by the use of Contracts?
- Is the interaction of Contracts with virtual functions, dynamic dispatch, and calls to function pointers and pointers to member functions as specified by the TS implementable and useful? What changes are desirable to improve the specification of Contracts in these regards?

Proposed Schedule

We believe a tight schedule is essential to achieve a lightweight process with minimal distraction to other working groups; otherwise, the exercise of producing the TS will interfere with its goal of efficient and rapid, useful feedback.

- **2022 November meeting (Kona)**:
  SG21 forwards design of Contracts TS to EWG.
- **2023 February meeting (location TBD)**:
  C++23 DIS is balloted.
  EWG forwards Contracts TS wording to Core.
- **2023 summer meeting (location and time TBD)**:
  Core approves Contracts TS wording; Contracts TS is balloted.
- **2023 autumn meeting (location and time TBD)**:
  C++23 is about to be published.
  Contracts TS ballot resolution takes place.
  Contracts TS is forwarded to ISO for publication.

Conclusion

The enormous effort made in previous attempts to adopt support for Contracts into the language makes clear that this feature is ardently desired. Participants in SG21 have brought together various viewpoints on distinct aspects of the problem, but data from real-world use of standardized Contracts by a wide audience is lacking. Let's allow previous success with the Coroutines, Modules, and Concepts TSs to guide us toward achieving similar success with Contracts.

Again, this proposal is intended to complement and inform SG21's work, not to compete with or bypass it. The proposed TS will provide valuable feedback from real-world experience with adoption of a truly useful product, and this feedback will assist SG21 in reaching consensus regarding timely incorporation into the International Standard. The risk of releasing the proposed TS is low, and the benefits for the wider C++ community are potentially vast.

Appendix: Why a TS *Is* Appropriate for Contracts

The direction group maintains [P2000R3] as the document providing guidance for the evolution of the language. Subsection 7.3 describes the role TSs serve and a list of questions that should essentially be answered by any TS proposal. We have copied that list and pasted it below, providing answers to each question.

- **Use TSs for library components.**

  Not applicable

- **Don't use TSs for a language feature unless the feature is a mostly self-contained unit.**

  We believe the proposed TS is entirely self-contained.

- **Never use a TS simply to delay; it doesn't simplify later decision making.**

  The goal of this TS is specifically to answer essential relevant questions within the timeframe of the next Standard and thereby *speed* adoption of this feature, not delay it.

- **When proposing a TS, specify the "aim": what the TS is supposed to learn or achieve.**

  The aim is to mitigate technical risk to the Minimum Viable Product (MVP) under development in SG21 so that it can land with confidence even late in the C++26 design cycle.

- **List "exit criteria" (TS to IS or whatever target) to allow people to determine whether the work is complete and whether it succeeded.**

  The work is complete when SG21 lands a Contracts feature into the C++ working draft. This TS is expected to *inform* that work, not *be* that work.

- **Consider other vehicles such as SG (Study Groups), IS, and not just TS.**

  The ongoing work in the current study group is proceeding more slowly than anticipated. The proposed TS will provide practical feedback in time to be useful. Moreover, much of the implementation work for the TS is already complete in various compilers but, for procedural reasons

only, is blocked from general release until a TS codifying the intended behavior is approved by the committee.

- **Consider some or all the following incomplete list of frequently asked questions in your deliberations and TS proposal and record their answers along with the aim and exit criteria:**
  - **Is there an implementation?**

    Yes, but by policy, that implementation cannot ship in current compilers in the absence of a TS
  - **Is it a Library or Language proposal, or does it involve both aspects?**

    Language, with a tiny library portion consistent with other core features having small library parts, such as runtime type identification and brace initialization.
  - **Is the proposal a foundational proposal, meaning many other C++ aspects/proposal depend on it, and/or it depends on many other C++ aspects/proposals?**

    Nothing in C++23 or C++26 is anticipated to depend on this language feature, nor does this TS depend on other proposals; all necessary and interesting language features with which this feature might interact are landed in C++23. This TS does, however, inform the common design and implementation space of a number of proposals competing with each other. This TS is not intended to compete directly with such proposals but instead to help advance aspects of them, independently of the TS itself.
  - **Is it independent of aspects of the language?**

    Yes, it is generally independent of other aspects of language design.
  - **Are there competing design proposals?**

    There are multiple competing design proposals, which is why progress on the feature has been blocked for so long. This TS will provide field experience to answer questions common to many, perhaps all, proposals in this space.

– **Is the proposal [so] complicated or large that you fear there will be error in design decision?**

  No. Although the design space is large, and has many competing perspectives, our concern is that most discussion has focused on where such designs diverge; without field experience, we risk missing basic flaws in our foundations.

– **Is it a research idea?**

  No. All the ideas present have been approved by EWG with the superconsensus mandate to make a change during ballot resoluton.

– **Is there substantial invention?**

  No. All the ideas present have been approved by EWG with the superconsensus mandate to make a change during ballot resoluton.

– **Can it be staged?**

  This is the staging part, to inform the IS process.

– **Is there a subpart that deserves to be in IS?**

  The goal is to reinforce landing parts already in progress into the current IS, not to directly replace them with this work.

– **Is the wording complicated or unconventional?**

  Mostly no. However, we are specifically seeking feedback on how the Standard specification can respond to a feature that is expressed through compiler switches. The TS will provide critically important feedback to address those concerns in the regular SG21 process for the proposed MVP, which is aimed at C++26.

– **Will the proposal benefit from early integration (can be applied to a WP)?**

  No. This proposal is not intended to land in the WP but instead provide early feedback to guide the existing *de novo* work for the next Standard.

– **Will you get feedback/testing only after TS publication or IS publication?**

  We expect the majority of feedback after an implementation becomes publicly available; said availability is blocked on the

publication of the TS.

– **Is there a motivation to slow down a proposal?**

  No! This TS is motivated by a strong desire to *accelerate* the adoption of the feature into the WP by gaining feedback to unblock existing proposals within the timeframe of the next Standard.

– **What would it take to turn the TS into an IS?**

  Consensus, but we do not expect to adopt the feature set of this TS wholesale. Instead we expect practical experience gained from publishing this TS will quickly help us to reach consensus in the ongoing feature development within SG21.

– **Are you juggling a large number of related or dependent proposals (other proposals that depend on this proposal)?**

  No. There are a number of proposals competing in this space, but the intent of quickly publishing this TS is to gain useful feedback that will inform *all* of them.

– **Are you aiming for user feedback?**

  Yes. We have an implementation ready for review in a major compiler so that early adopters can give urgently needed critical feedback in time for developing the MVP of the feature for C++26.

– **Are you aiming for implementation feedback?**

  Yes, there are a variety of implementation specific questions, notably around ABI.

– **Is there a scheduling concern to make C++xx for it or its dependents?**

  Yes! The scheduling concern is to ship early enough to provide feedback within the next Standard development cycle. The implementation of the feature in at least one major compiler is largely complete, making a quick delivery practicable, but if the TS does not proceed promptly toward publication, it will have missed its window to provide useful feedback in time to publish an MVP for the feature in C++26.

References

[N1773] David Abrahams, Lawrence Crown, Thorsten Ottosen, and James Widman, *Proposal to add Contract Programming to C++ (revision 2)*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1773.html

[N4075] John Lakos, Alexei Zakharov, and Alexander Beels, *Centralized Defensive-Programming Support for Narrow Contracts (Revision 6)*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4075.pdf

[P0542R5] G. Dos Reis, J. D. Garcia, J. Lakos, A. Meredith, N. Myers, and B. Stroustrup, *Support for contract based programming in C++*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0542r5.html

[P1323R2] Hubert S.K. Tong, *Contract postconditions and return type deduction*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1323r2.html

[P1344R1] Nathan Myers, *Pre/Post vs. Expects/Ensures*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1344r1.md

[P1607R1] Joshua Berne, Jeff Snyder, and Ryan McDougall, *Minimizing Contracts*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1607r1.pdf

[P1680R0] Andrew Sutton and Jeff Chapman, *Implementing Contracts in GCC*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1680r0.pdf

[P1823R0] Nicolai Josuttis, Ville Voutilainen, Roger Orr, Daveed Vandevoorde, John Spicer, and Christopher Di Bella, *Remove Contracts from C++20*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1823r0.pdf

[P2000R3] H. Hinnant, R. Orr, B. Stroustrup, D. Vandevoorde, M. Wong, *Direction for ISO C++*
https://open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2000r3.pdf

[P2521R2] Gašper Ažman, Joshua Berne, Bronek Kozicki, Andrzej Krzemieński, Ryan McDougall, Caleb Sunstrum, *Contract support — Working Paper*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2521r2.html

[P2660R0] Brian Bi, *Proposed Contracts TS*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2660r0.pdf

[P2661R0] Brian Bi, *Miscellaneous amendments to the Contracts TS*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2661r0.pdf