P2900 Is Superior to a Contracts TS

Document #:	P3276R0
Date:	2024-05-17
Project:	Programming Language C++
Audience:	SG21 (Contracts)

Reply-to:	Joshua Berne <jberne4@bloomberg.net>
	Steve Downey <sdowney2@bloomberg.net>
	Jake Fevold <jfevold@bloomberg.net>
	Mungo Gill <mgill83@bloomberg.net>
	Rostislav Khlebnikov <rkhlebnikov@bloomberg.net>
	John Lakos <jlakos@bloomberg.net>
	Alisdair Meredith <ameredith1@bloomberg.net>

Abstract

It has been proposed that the Contracts facility being produced by SG21 should ship as a TS rather than as part of the IS. While some technical and political concerns with P2900 need to be considered, a TS is a viable solution when only it, not existing proposals, has the potential of answering those questions. Our position is that none of the current issues fit into this category, and we strongly believe that work on evolving P2900 should continue until it has consensus to be adopted into the IS, preferably in the C++26 timeframe.

Contents

Revision History

Revision 0

- Original version of the paper

1 Introduction

Bloomberg has been actively investing to introduce a contract-checking facility into C++ for many years and has produced [N4378], [P0542R5], [P1429R3], [P1607R1], [P2900R6], [P2755R1], and many additional papers to achieve that goal. It has made this investment because it believes that a contract-checking facility is the single most powerful tool that can be added to the language to improve the correctness — and, therefore, safety — of both existing and future C++ code.

To that end many Bloomberg and non-Bloomberg engineers have, since the inception of SG21, worked tirelessly to produce a Contracts facility that has both maximal consensus and the potential to provide solutions for the many use cases for Contracts that have been identified (see [P1995R0]). As part of that ongoing work, SG21 had very strong consensus to pursue the plan laid out in [P2695R1], focusing first on producing and releasing a viable foundation for Contracts in C++ that can eventually evolve to support the full breadth of known use cases. We believe that this approach will lead to an incredible outcome for C++ and fits perfectly with the already proven successful release model, as laid out in [P1000R5], that WG21 has followed consistently since C++11. In particular, by including the parts of the feature that are ready when the Standard is set to ship, we deliver real utility to our users and then continue to build on that foundation with future iterations.

Recently, Ville Voutilainen proposed in [P3265R0][1] that WG21 should be targeting a technical specification (TS) as an initial vehicle for Contracts. As described in [SD4], a technical specification is an experimental branch of the International Standard (IS) for a feature that does not yet have consensus to be included in the IS. An important part of the established process for a TS is to list specific goals for that TS. Importantly, those goals must contain both what knowledge will be gained from the TS as well as exit criteria under which the TS will be merged into the IS (or, possibly, abandoned).

To be successful, any plan to ship as a TS must be an improvement over the current plan in [P2695R1], which aims to answer all questions needed before inclusion of Contracts in the IS via a paper produced by SG21 as a group — that paper is the Contracts MVP, currently [P2900R6]. Of the open issues of which we are aware, we believe those issues can be addressed by continued work on [P2900R6] and that no clear explanation as to *how* the TS improves our ability to answer those questions has been offered.

[1]Note that the TS proposal currently being considered is not to be confused with the TS proposal that Bloomberg put forward in 2022, [P2659R2]. That proposal had specific goals aimed at improving the quality of the SG21 Contracts proposal. Discussions of that proposal clarified to us that the existence of a TS did not materially impact our ability to answer the questions we had at the time, and we, along with the rest of SG21, realized that pursuing a TS at the time offered no benefit and had no consensus.

2 Questions a TS Might Address

Let's consider the various design issues related to Contracts, how we expect SG21 will address them using further iterations of [P2900R6], and why we believe a TS will not facilitate dealing with those issues.

- **Implementation Experience** — Implementation experience requires a robust specification and investment into producing implementations. Open-source implementations of C++20 contracts, which never had a TS, have already been released in GCC. Bloomberg is in the process of continuing those efforts to implement the Contracts MVP in GCC and is beginning efforts to see a `clang` implementation made available.

  Having multiple open source implementations, especially implementations that share a platform and must remain ABI compatible with one another, is certainly a meaningful indication that the SG21 proposal for Contracts is implementable and usable. A TS will not change this conclusion. During the discussions of the previous Contracts TS proposal (2022-12-01 Contracts Telecon), Microsoft indicated that they would not implement Contracts even if it were a TS, and we have seen no reason to believe that their position will change, given ongoing Microsoft objections to SG21's consensus as laid out in [P3173R0].

  Perhaps some members think that an implementation *based on a paper* will not be merged into open-source implementations yet one *based on a TS* might have a greater chance of adoption; this difference is moot since, currently, compiler branches of all open-source compilers for experimental papers are frequently made available through Compiler Explorer (`https://godbolt.org`) and readily locally buildable for experimentation. More importantly, at least for GCC, a willingness to merge support for Contracts into the release build has already been shown, and a TS was never part of that plan.

- **Field Experience** — Generally and in our experience, production systems do not make use of compiler extensions, whether they are implementations of a TS or a paper. For those organizations that do wish to make use of a feature, again, the text on which a compiler extension is based is significantly less relevant than the robustness of that specification and the implementation in question.

  In practice, field experience can come in a few forms.

  – Small-scale experimentation can happen as soon as implementations are available on Compiler Explorer, which clearly does not depend on a TS.

  – Large-scale experimentation, where existing projects are tested with the use of Contracts, is equally feasible as soon as implementations are available. Bloomberg did such experiments with the GCC implementation of C++20 contracts long before they were available in the release build of GCC.

  – Production use of the Contracts facility in new codebases built from the start to use it depends on a number of things. Implementations must be available and released, and organizations must be willing to use such experimental features as a foundation for new large-scale endeavours. We expect this sort of experience to take time and to be unaided by whether Contracts is a TS or part of the IS.

Therefore, barring concrete promises from significant userbases indicating otherwise, we expect Contracts to garner the same amount of field experience — irrespective of whether Contracts is based on a TS or a paper — once implementations are available.

- **WG21 Consensus on Design** — From the presentation of [P2900R6] to EWG, we can conclude that the significant work that has been done to motivate the design of the proposal and to achieve consensus in SG21 for its contents must be repeated in EWG. Many of the initial responses — and corresponding votes — seen in EWG were opinions that were voiced, discussed, and resolved in SG21 during the long process of coming to consensus on the Contracts MVP. The complexity in the Contracts MVP and the decisions it makes are a natural result of aiming to cover the use cases that a mature and comprehensive Contracts facility must satisfy.

  In addition, EWG's consensus on a design is no less a requirement for a TS than it is for the IS. Any agreement reached regarding what to include in a Contracts TS based on the proposal from SG21 would also apply to what to include in the IS. If we set a lower bar for what should be included in a TS, then we must also provide a mechanism for deciding when consensus is reached to raise the bar to the level of the IS; we have seen no such procedure described by proponents of a Contracts TS.

- **Coherence and Coverage** — Using a TS for Contracts will not equate to immediate support for features that are not currently part of the Contracts MVP. Those features might be incorporated into the TS during its lifetime, but the same can be said of incorporating those features into [P2900R6] during its lifetime.

  Regarding how to handle virtual functions, existing proposals ([P3097R0] and [P3165R0]) are already gaining consensus in SG21 and could easily be incorporated into [P2900R6] once consensus is achieved or could be added later if they need more time to mature.

  In the case of function pointers or coroutines, we currently lack any comprehensive proposals that answer all questions, nor do we have thoroughly considered implementation strategies for either. Papers are in progress for these issues, but waiting on their availability simply means waiting indefinitely to have Contracts in the IS because we insist on supporting arguably niche use cases.

  In all of these cases, however, any proposal — once it has consensus and is available — will be equally ready for integration into the Contracts MVP, a TS, or the IS. We do not see any advantage in having a TS for this purpose over continuing the current path with the Contracts MVP.

- **Increased Safety** — Much time has been spent discussing the use of a strictly safe subset of the language in contract assertion predicates. We do not believe that such a safe subset is viable for use in Contracts, nor do we believe that waiting until that subset reaches sufficient maturity is an effective use of time. Insisting that we move all future standardization efforts to the act of standardizing a provably safe language in lieu of extending and evolving C++ naturally is, to us, equivalent to insisting that we halt all forward progress on the language. This myopic mindset does not serve our users, our business, or our long-term need for a qualitatively safer programming language.

We fully support and have invested heavily in such exploration into making C++ safer. We think such efforts must be fully applicable to new code, to existing code, to contract assertions predicates, and to expressions anywhere else in the language; otherwise, both those safety features and contract assertions will become toy languages that see minimal real-world use.

3 The Expected Cost of a TS

Shipping a feature in a TS brings with it a number of procedural overhead costs that are nonobvious and that seem to only further delay progress.

1. A TS must be based on a *published* version of the Standard, whereas a paper, such as [P2900R6], can contain wording relative to the current *draft* Standard. As of the time of this paper's writing, a TS would require rebasing [P2900R6] onto the C++20 Standard. Hopefully, such a TS would soon be rebaseable onto the C++23 Standard, but when that could happen is still an open question. To maximize the ability to keep the TS valid through its lifetime, we might even need to wait for the C++26 Standard to become available, which could be delayed as far as 2027.

2. To keep the TS in a state in which it could feasibly be merged once it does reach consensus, work on the wording in the TS must be doubled, making sure it is correct with respect to the Standard's published basis *and* its current draft. The older the published basis for the TS is, the more cumbersome this maintenance work becomes and the more a TS is an obstacle to progress. None of that duplicated effort will realize any benefits in the long-term quality of the C++ language.

3. Discussion time for a TS in evolution groups will, inevitably, be prioritized below discussion time for any changes targeting the IS itself. As a result, any reasonable estimate of how long even preparing a TS for publication would take must include multiple in-person meetings in which only minimal time will be allocated, and we already know that achieving consensus will require significant effort.

Any consideration of shipping Contracts in a TS must question not just whether the TS will facilitate resolving the issues it lists, but whether the significant added costs make that effort worthwhile.

4 Conclusion

We reiterate that we have seen no evidence that shipping a Contracts TS will result in answering any of the open questions that must be resolved before we can achieve consensus to adopt Contracts into the C++ Standard. Overall, we have seen indication that shipping as a TS would only delay delivery of a feature that is essential for correctness and safety to C++ users around the world. Such a delay would imply to governments, regulatory bodies, and users that we do not actually take language safety issues seriously and are not working to resolve such issues as quickly as possible.

Our opinion on this process can most certainly be swayed, but we have yet to see a proposal for a TS that follows the expected guidelines, i.e., that clearly states the following:

- The questions that releasing a TS intends to answer.

- How those questions will be answered by a TS.
- How those questions could not be answered by continuing work on P2900.
- What the exit strategy from such a TS which results in a Contracts Facility for C++ being adopted would be.

Acknowledgements

Thanks to Lori Hughes for ensuring that the words in this paper remain readable and clear.

Bibliography

[N4378] John Lakos, Nathan Myers, Alexei Zakharov, and Alexander Beels, "Language Support for Contract Assertions", 2015
http://wg21.link/N4378

[P0542R5] J. Daniel Garcia, "Support for contract based programming in C++", 2018
http://wg21.link/P0542R5

[P1000R5] Herb Sutter, "C++ IS schedule", 2023
http://wg21.link/P1000R5

[P1429R3] Joshua Berne and John Lakos, "Contracts That Work", 2019
http://wg21.link/P1429R3

[P1607R1] Joshua Berne, Jeff Snyder, and Ryan McDougall, "Minimizing Contracts", 2019
http://wg21.link/P1607R1

[P1995R0] Joshua Berne, Andrzej Krzemieński, Ryan McDougall, Timur Doumler, and Herb Sutter, "Contracts - Use Cases", 2019
http://wg21.link/P1995R0

[P2659R2] Brian Bi and Alisdair Meredith, "A Proposal to Publish a Technical Specification for Contracts", 2023
http://wg21.link/P2659R2

[P2695R1] Timur Doumler and John Spicer, "A proposed plan for contracts in C++", 2023
http://wg21.link/P2695R1

[P2755R1] Joshua Berne, Jake Fevold, and John Lakos, "A Bold Plan for a Complete Contracts Facility", 2024
http://wg21.link/P2755R1

[P2900R6] Joshua Berne, Timur Doumler, and Andrzej Krzemieński, "Contracts for C++", 2024
http://wg21.link/P2900R6

[P3097R0] Timur Doumler, Joshua Berne, and Gašper Ažman, "Contracts for C++: Support for virtual functions", 2024
http://wg21.link/P3097R0

[P3165R0] Ville Voutilainen, “Contracts on virtual functions for the Contracts MVP”, 2024
http://wg21.link/P3165R0

[P3173R0] Gabriel Dos Reis, “<a href="../2024/p2900r6.pdf">P2900R6</a> may be minimimal, but it is not viable”, 2024
http://wg21.link/P3173R0

[P3265R0] Ville Voutilainen, “Ship Contracts in a TS”, 2024
http://wg21.link/P3265R0

[SD4] “Practices and Procedures: The "How We Work" Cheat Sheet”
https://wg21.link/sd4