Do Not Ship Contracts as a TS

Timur Doumler (papers@timur.audio)
John Spicer (jhs@edg.com)

Document #: P3269R0
Date: 2024-05-21
Project: Programming Language C++
Audience: SG21, EWG

1 Introduction

By approving the roadmap proposed in [P2695R1] with very strong consensus, SG21 (Contracts Study Group) committed to deliver a Contacts MVP proposal for C++26. Since then, huge progress toward this proposal has been made, culminating in SG21 forwarding the Contracts MVP paper [P2900R6], once again with very strong consensus, to EWG for design review.

The EWG design review of [P2900R6] at the March 2024 WG21 meeting in Tokyo revealed a sustained opposition from one vendor to a few concrete aspects of the design (see [P3173R0]), and a lack of consensus in EWG on a few aspects, with some overlap between these two sets of concerns. Some of these concerns have since been addressed for the upcoming pre-St. Louis revision [P2900R7], and others are currently in the process of being discussed and addressed in SG21 and SG23 (see [P3197R0] for the list of concerns and a roadmap for addressing them).

In the wake of these discussions, [P3265R0] recently suggested that the Contracts MVP proposal should no longer target the C++26 IS but instead a Contracts TS.

We, the chairs of SG21, recommend strongly against targeting a Contracts TS.

2 Why we need Contracts in the IS

2.1 Contracts address an important part of the C++ safety problem

The biggest challenge that the C++ programming language faces today is a lack of safety and security, caused by the ease of inadvertently writing code that has unbounded undefined behaviour. The purpose of adding contract assertions to the C++ language is to make writing safe code easier. In fact, contract assertions might be the most effective way of reducing undefined behaviour in production that we currently have. Out-of-bounds access into a container is just one example of a frequently exploited vulnerability that can be easily mitigated with Contacts. As such, the Contracts facility needs to be added to the language as soon as possible. It is unfortunate enough that Contracts missed the train for C++20 and

again for C++23. We should do everything we can to preserve the realistic chance to include Contracts in C++26. Prematurely discarding that chance would do significant damage to the C++ ecosystem.

2.2 The Contracts MVP provides a useful starting point

The purpose of the Contracts MVP is to have, in the IS, an initial set of functionality upon which WG21 can agree and that can be extended later. This MVP is supposed to provide the core functionality for a Contracts facility, a platform that we can extend in the future. The Contracts MVP — by definition of *minimal* viable product — cannot satisfy everyone's use cases but is designed to satisfy *basic* use cases from the start and simultaneously to not preclude satisfying anyone's use cases in the future via post-MVP extensions. This extensibility and foresight regarding future directions is the core design principle of [P2900R6]. Getting the Contracts MVP into C++26 will therefore achieve what a Contracts TS might achieve and do so much more effectively.

2.3 The Contracts MVP is not a compromise on quality

The urgency of having Contracts in the IS notwithstanding, to be added to the C++ Standard, a Contracts facility, like any new language feature, needs to meet a high bar on both consensus and the level of confidence that we do not ship anything broken. If either bar is unmet, we need to do the work required to meet that bar. If that work is not complete in time for C++26, then Contracts will unfortunately have to wait for C++29. We are confident that we can complete the MVP in time for C++26, but if we are mistaken, we believe that the goal should then be to add Contracts to the C++29 Working Draft as early in the process as possible, instead of pursuing a TS. The Contracts MVP is intended to be a compromise on the feature set — *minimal* in the sense that it represents the minimal set of features upon which WG21 can achieve consensus — but not a compromise on quality.

3 Why a Contracts TS does not help

3.1 A TS would be an inefficient use of time and resources

The TS process is slow and bureaucratic. A TS must be approved by WG21 plenary, go through a national ballot, go through NB comment resolution, be approved by WG21 plenary again, and then be published by ISO. This process takes about a year and consumes a significant amount of precious committee resources — resources that can be utilised more efficiently by focusing on fixing, *now*, the actual remaining technical issues and areas of disagreement in [P2900R6]. About a year remains before the cutoff date for adding the feature to the C++26 Working Draft. This is enough time, and we should use it wisely. Spending time on considering a Contracts TS is an unnecessary distraction.

3.2 A TS is not needed to get implementation experience

It is sometimes said that a TS serves to provide implementation experience for an experimental feature. However, for a feature as important for the C++ ecosystem as Contracts, this argument is a red herring. We do not need a TS to gain implementation experience; we just need a specification of reasonable quality (which [P2900R6] provides)

and someone willing to invest the time and money to implement that specification. Whether that specification has been published as a TS does not matter as long as there is enough interest in the feature itself. The Contracts facility itself is proof that a TS is not needed: its previous incarnation, [P0542R5], also known as *C++2a Contracts*, has been implemented by GCC, and that implementation has proven to be a valuable source of information, although C++2a Contracts never shipped as a TS.

3.3 A TS would not help answer the outstanding issues

The main reason for publishing a feature in a TS is to help answer open design questions about the feature. According to [SD-4], before a TS can be published, the specific questions that the TS is supposed to answer must be listed. What are we hoping to learn through this TS? What are the exit criteria and, especially, the list of must-address controversial issues before we can consider merging this TS into the IS?

However, upon closer examination, the current controversies over [P2900R6] are all either technical issues that can be resolved relatively easily in the remaining time for C++26 and, therefore, do not require a TS cycle or more fundamental disagreements that a TS would not in any way help resolve. In the following section, we briefly discuss all of these issues.

4 Outstanding issues with the Contracts MVP

4.1 Virtual functions

[P3173R0] states that, to be viable, a Contracts MVP needs to support contract assertions on virtual functions. We have mature proposals including wording ([P3097R0] and [P3165R0]) on the table to add this feature. These proposals need to be discussed and approved by SG21, merged into the Contracts MVP, and reviewed by EWG. A prototype implementation of this feature can be developed in parallel. We estimate that this work can be accomplished in far less time than that required to ship a TS, and we do not see any benefits that such a TS would provide on this topic.

4.2 Function pointers

The Contracts MVP already guarantees that the contract assertions of a function are checked even if it is called indirectly through a function pointer. However, we do not have the ability to add separate, different contract assertions to the function pointer itself. [P3173R0] seems to suggest that, to be viable, a Contracts MVP needs to provide this functionality, yet we currently do not know how to achieve this in any meaningful way.

Contract assertions appertaining directly to a function pointer would have to become either part of the function type or part of the pointer value; neither approach fits into the current C++ model of how function pointers work. [P3173R0] offers no solutions to this question. [D3271R0] explores an alternative design that creates the new notion of a *function usage type* rather than allowing contract assertions on function pointers.

We believe that such a feature can be added post-MVP if and when an appropriate design materialises and is not a requirement for the MVP; most of EWG seems to agree that support for function pointers in the MVP is not essential (see [P3197R0]). We do not see how pursuing a TS now could contribute anything here.

4.3 Coroutines

For coroutines, assertion statements are already supported, but we do not know how to add meaningful support for precondition and postcondition specifiers. No design has been proposed for specifying contract assertions on the entry and suspension points of the coroutine itself. [P2957R1] proposes to add contract assertions to the so-called "ramp function" that creates the coroutine; however, in this scenario, postconditions do not work and the usefulness of preconditions is limited. In any case, SG21 as well as the authors of [P2957R1] themselves consider this to be a post-MVP extension rather than an MVP feature. Most of EWG seems to agree that support for coroutines in the MVP is not essential (see [P3197R0]). As with function pointer support, we do not see how pursuing a TS now could contribute anything here.

4.4 Elision, duplication, and `const`-ification

The questions of elision, duplication, and `const`-ification of predicate evaluations have been much debated in recent weeks ([P3228R1], [P3257R0], [P3264R0], [P3268R0], [P3281R0], [D3270R0]). Upon closer examination, only a limited number of possible solutions exist, none of which seem to be right or wrong per se — they simply favour different use cases.

We must decide which use cases we want to prioritise over others, and the correct solution will then emerge. This work is already under way and can likely be completed in far less time than what would be required to ship a TS. We do not see any benefits that such a TS would provide on this topic.

4.5 Throwing contact-violation handlers

The question of whether to allow throwing contract-violation handlers is similar to the question of elision, duplication, and constification. The two possible options are, on an implementation that supports user-defined throwing violation handlers, to allow such a handler to exit via an exception or to require the handler to be `noexcept` on all implementations. Both options have different tradeoffs. Allowing throwing from a contract-violation handler enables use cases that some consider critically important for Contacts to support but have implications for exception-unsafe code and the broader use of `noexcept` that some consider unfortunate.

We need to discuss these tradeoffs with WG21 and then get consensus on a solution. This is by no means an insurmountable obstacle; however, shipping Contracts in a TS now will not help resolve it.

4.6 Contracts in the C++ standard library

Usage experience with contract assertions in an *implementation* of the C++ standard library is something that a major compiler vendor is already committed to providing, based on the

specification in [P2900R6], regardless of whether the Contracts facility ships as a TS; a TS is therefore unnecessary to achieve this goal.

If one is interested in what adding Contacts to the *specification* of the C++ standard library might look like, relevant material is readily available: [P2755R1] section 4.4 provides a preview of how adding contract assertions to the public functions of `std::vector` (an archetypical example of an STL container) might appear, and [D3212R0] is doing the same for `std::sort` (an archetypical example of an STL algorithm). More such case studies can be done if desired.

Note, however, that adding contract assertions to the specification of the C++ standard library has not actually been proposed, and the current consensus in SG21 is that doing so is not desirable: for a number of reasons, implementations should be allowed but *not* required to use contract assertions to check the preconditions on standard library functions defined in the C++ Standard.

4.7 Contracts and undefined behaviour

Contract assertions can be affected by undefined behaviour in two often conflated but completely orthogonal ways; they are described in detail in [P2900R6], Section 3.4.6.

First, a contact check evaluated with the *observe* evaluation semantic (i.e., a contract check that continues execution after a contract violation) might be time-travel-optimised out of existence if undefined behaviour occurs some time *after* the contract check. This issue can be fixed by specifying that a contract assertion acts as an optimisation barrier ([P1494R2]).

Second, undefined behaviour can occur *during* a contract check because contract predicates are C++ expressions and, as such, follow the usual rules for expression evaluation. Therefore, evaluating a contract predicate can directly or indirectly lead to undefined behaviour, for example via signed integer overflow or dereferencing of an invalid pointer (possibly inside some other opaque function).

It has been proposed in [P2680R1] (see also [P2700R0] and [P3285R0]) that instead of following the usual rules for expression evaluation, contract predicates should, by default, follow different, restricted rules (so-called "non-relaxed contracts") that limit the possibility of undefined behaviour. SG21 and SG23 have spent much time discussing this direction; both groups decided against pursuing it further and reached consensus that the current treatment of undefined behaviour during contract checks in [P2900R7] is sufficient.

It is unclear how to specify such an alternative set of rules for C++ expression evaluation, how many cases of undefined behaviour it could actually prevent, and whether the resulting subset of C++ expressions would be expressive enough to be of any practical use for writing contract predicates. Pursuing a Contracts TS now will not help answer any of these questions. Holding up the standardisation of a Contracts facility — which in itself goes a long way toward eliminating undefined behaviour from C++ programs — to wait for the invention of such an alternative set of rules for C++ expression evaluation would, in our opinion, be a very serious mistake. The problem with contract assertions is not that they are not safe; the problem is that they are not *there*.

Summary

Our position as the chairs of SG21 is that targeting a Contracts TS would be a mistake. We recommend that the current Contracts MVP proposal [P2900R6] continue to target the C++ IS.

Though some unresolved questions regarding [P2900R6] are being actively discussed, a Contracts TS would not contribute anything meaningful toward resolving these questions and would unnecessarily delay progress. Further, while it would undoubtedly be useful to gain implementation experience with this feature, a Contracts TS is not required in order to do so.

If we focus — *now* — on resolving the open questions in the Contracts MVP, we believe that consensus can be achieved much sooner than if we invested time in the process to publish a Contracts TS. Continuing to pursue the Contracts MVP for the IS would be a much more efficient use of committee time and resources to achieve this consensus. The open questions are not an insurmountable obstacle.

The biggest challenge for the future of C++ is to make it a safer language. WG21 has a responsibility to address this challenge. C++ would be significantly safer with a Contracts facility than it is without. We should, therefore, focus on adding such a facility to the C++ IS as early as we can — in the C++26 cycle if at all possible, or otherwise early in the C++29 cycle.

References

[D3212R0] Andrzej Krzemieński (2024-03-29). The contract of `sort()`.

[D3270R0] John Lakos and Joshua Berne (2024-05-17). Repetition, Elision, and `const`-ification with Regard to `contract_assert` – A Principled-Design Analysis.

[D3271R0] Lisa Lippincott (2024-05-08). Function Usage Types (Contracts for Function Pointers).

[P0542R5] Gabriel Dos Reis, J. Daniel Garcia, John Lakos, Alisdair Meredith, Nathan Myers, and Bjarne Stroustrup (2018-06-08). Support for contract based programming in C++.

[P1494R2] Davis Herring (2021-11-10). Partial program correctness.

[P2680R1] Gabriel Dos Reis (2022-12-15). Contracts for C++: Prioritizing Safety.

[P2695R1] Timur Doumler and John Spicer (2023-02-09). A proposed plan for Contracts in C++.

[P2700R0] Timur Doumler, Andrzej Krzemieński, John Lakos, Joshua Berne, Brian Bi, Peter Brett, Oliver Rosten, and Herb Sutter (2022-11-29). Questions on P2680 "Contracts for C++: Prioritizing Safety".

[P2755R1] Joshua Berne, Jake Fevold, and John Lakos (2024-04-11). A Bold Plan for a Complete Contracts Facility.

[P2900R6] Joshua Berne, Timur Doumler, and Andrzej Krzemieński (2024-02-29). Contracts for C++.

[P2900R7] Joshua Berne, Timur Doumler, and Andrzej Krzemieński (2024-05-22). Contracts for C++.

[P2957R1] Andrzej Krzemieński and Iain Sandoe (2024-01-13). Contracts and coroutines.

[P3097R0] Timur Doumler, Joshua Berne, and Gašper Ažman (2024-04-12). Contracts for C++: Support for Virtual Functions.

[P3165R0] Ville Voutilainen (2024-02-26). Contracts on virtual functions for the Contracts MVP.

[P3173R0] Gabriel Dos Reis (2024-02-15). P2900R6 May Be Minimal, but It Is Not Viable.

[P3197R0] Timur Doumler and John Spicer (2024-04-12). A response to the Tokyo EWG polls on the Contracts MVP (P2900R6).

[P3228R1] Timur Doumler (2024-05-07). Revisiting side effects, elision, and duplication of contract predicate evaluations.

[P3257R0] Jens Maurer (2024-04-26). Make the predicate of `contract_assert` more regular.

[P3264R0] Ville Voutilainen (2024-05-03). Double-evaluation of preconditions.

[P3265R0] Ville Voutilainen (2024-05-05). Ship Contracts in a TS.

[P3268R0] Peter Bindels (2024-05-07). C++ Contracts Constification Challenges Concerning Current Code

[P3281R0] John Spicer (2024-05-14). Contract checks should be regular C++.

[P3285R0] Gabriel Dos Reis (2024-05-15). Contracts: Protecting The Protector.

[SD-4] Herb Sutter (2023-07-01). WG21 Practices and Procedures.