C++ Contracts on Trial - Does P2900 Survive Cross-Examination?

Document Number:	P4208R0
Date:	2026-05-01
Intent:	Inform
Audience:	WG21

Reply-to:	Claude Opus 4.6 Vinnie Falco vinnie.falco@gmail.com

Table of Contents

Abstract

An AI examines P2900R14 "Contracts for C++" by applying the Advocatus Diaboli from P4207R0 "Prosecute Your Paper To Improve It." The Advocatus is a structured adversarial review tool for WG21 papers. It reads the paper, assembles a dossier, tests every claim, cross-examines its own findings, and delivers a formal verdict.

Revision History

R0: May 2026 (pre-Brno mailing)

- Initial version.

---

1. Disclosure

The author provides information and serves at the pleasure of the committee.

This paper, and every part of the process - building, writing, editing - was produced with the help of a large language model.

This paper asks for nothing.

Everything below this line is the output of a large language model. The better the model, the better the results. AI output must always be verified by a human.

2. Animadversiones of the Advocatus Diaboli

Opening Invocation

The Advocatus Diaboli is called to examine the cause of "Contracts for C++" (P2900R14).

Postulator: Joshua Berne, Timur Doumler, Andrzej Krzemienski. Audience: EWG. Nature of the cause: ask-paper.

Posture: defensio.

The tribunal convenes.

The Seal

Cum obiectionibus.

The Advocatus examined the cause across fourteen revisions, sixteen design principles, one hundred and nineteen pages of specification, and a public record spanning five years of SG21 deliberation, two years of EWG scrutiny, finalized CWG and LWG wording review, plenary adoption with strong consensus, seven national body comments requesting removal, three competing counter-proposals, and one set of published concerns from a co-author of the paper itself.

One objection survives. Eleven sections earn the *approbatio*. The cause proceeds.

Approbationes

The following sections and claims withstand adversarial examination. The Advocatus Dei prevailed in each case. Attacks on these positions would be self-defeating or would engage ground the paper has already fortified.

Approbatio 1: The Design Principles Framework (Section 3.1). Sixteen named principles govern the proposal's design. Each principle is stated, justified, and cross-referenced against specific design decisions. The principles are internally consistent: the Prime Directive (Principle 1) flows into the Redundancy Principle (Principle 2), which flows into the secondary principles on compile-time transparency (3--5), runtime purity (6--9), semantic integrity (10--12), extensibility (13--14), and compatibility (15--16). The Advocatus tested the internal consistency and found no contradiction. The framework withstands opposition because it provides a principled response to any design question: the answer traces back to a named principle. *This section withstands opposition.*

Approbatio 2: The Syntax Design (Section 3.2). The choice of `pre`, `post` as contextual keywords and `contract_assert` as a full keyword is well-motivated. Contextual keywords avoid breaking existing code that uses `pre` or `post` as identifiers. The full keyword for `contract_assert` is necessary for disambiguation and is sufficiently unlikely to conflict with existing identifiers. The placement of function contract specifiers after trailing return types and requires clauses follows a consistent grammar. The Advocatus filed a candidate charge against the length of `contract_assert`; the Dei dismissed it under Dignitas. *This section withstands opposition.*

Approbatio 3: The Four Evaluation Semantics (Section 3.5.4). The semantics form a coherent lattice: ignore (no checking), observe (check and continue), enforce (check and terminate), quick-enforce (check and terminate without handler). Each serves a distinct deployment scenario. The Tokyo EWG poll on enforce-only was rejected 6--1--3--15--24, confirming broad committee support for the four-semantic design. The observe semantic enables gradual deployment; quick-enforce enables minimal-footprint safety-critical builds; ignore enables maximum-performance release builds. The Advocatus tested whether the four semantics were over-specified or redundant and found each filling a distinct niche. *This section withstands opposition.*

Approbatio 4: Predicate Side-Effect and Elision Rules (Sections 3.5.7--3.5.8). The paper explicitly states that predicates may be evaluated zero, one, or many times. It demonstrates the consequences with concrete examples (`i may be 0, 1, 17, etc.`). It explains WHY: caller-side and callee-side checking may both occur, and optimizer freedom enables proof-based elision. The paper does not hide behind this permission --- it warns users against depending on side effects. The Advocatus filed a candidate charge that this surprised reasonable users; the Dei dismissed it under Confessio. The paper confesses the consequence in full, with examples and warnings. Attacking this position requires arguing that the paper should have forbidden side effects entirely, which would break existing patterns and constrain future evolution. *This section withstands opposition.*

Approbatio 5: Observable Checkpoints (Section 3.5.3). The paper addresses the interaction between contract assertions and undefined behavior directly. By making the beginning of contract predicate evaluation an observable checkpoint, the paper prevents time-travel optimizations from eliding contract checks that precede UB. The mechanism is built on [P1494R5] and is precisely scoped: it mitigates "certain cases" and does not overclaim. The paper also explains what it does NOT protect against (UB inside the predicate). The Advocatus tested whether the mitigation was sufficient and found the claims accurately scoped. *This section withstands opposition.*

Approbatio 6: Coroutines Treatment (Section 3.5.2). The paper applies contracts to coroutines by treating the ramp function as the contractual boundary. This follows from the fundamental principle that "the coroutineness of a function is an implementation detail." Preconditions are evaluated before the ramp function body; postconditions are evaluated when the ramp function returns. The paper explicitly acknowledges the limitation that postcondition assertions cannot odr-use nonreference parameters in coroutines (because coroutines may move from their parameters). Three alternative implementations are shown to demonstrate behavioral equivalence. The Advocatus filed a candidate charge that coroutine postconditions have limited utility; the Dei dismissed it under Confessio. *This section withstands opposition.*

Approbatio 7: Features Not Proposed (Section 2.3). The paper explicitly lists eleven features intentionally excluded from the MVP: virtual function contracts, function pointer contracts, postcondition old-values, the assume semantic, assertion levels, expression-level semantic specification, exception postconditions, unevaluable predicates, stateful assertions, invariants, and procedural interfaces. For each, the paper notes it was "in some shape or form, part of previous Contracts proposals." This section inoculates the paper against the most common attack --- "why didn't you include X?" --- by pre-answering the question. The Advocatus filed candidate charges on the absence of virtual function contracts and assertion levels; the Dei dismissed both under Confessio. The postulator has already surrendered this ground voluntarily, and charging a confession is theater, not prosecution. *This section withstands opposition.*

Approbatio 8: Implicit Const Design (Section 3.4.2). The paper applies implicit const to variables referenced within contract predicates to prevent accidental modification. It is transparent about the limitations: const is shallow and does not propagate through pointers. The paper provides code examples showing that `*p = 5` is permitted through a pointer while direct `x = 0` is ill-formed. The rationale is explicit: deep const would make raw and smart pointers behave differently, which is undesirable. National body comments ES55 and CZ58 challenged this design; P3846R0 Concern 8 provides a comprehensive response. The Advocatus filed a candidate charge on the inconsistency of shallow const; the Dei dismissed it under Confessio. The paper names the limitation before any critic can. *This section withstands opposition.*

Approbatio 9: ABI and Backward Compatibility (Principles 15--16). The paper claims that adding contract specifiers to an existing function preserves ABI backward-compatibility and does not break existing correct client code. These claims are verified by the design: contract assertions do not change function types, do not alter calling conventions, and contextual keywords do not conflict with existing identifiers. CWG wording review confirmed these properties. No counter-evidence exists in the positio. *This section withstands opposition.*

Approbatio 10: The Companion-Paper Architecture. The paper defers detailed motivation, history, alternative designs, and deployment experience to [P2899R1]. This is appropriate for a 119-page proposal that already contains design rationale, specification, and formal wording. The separation prevents the paper from becoming unwieldy while ensuring the rationale is on the record. The Advocatus tested whether essential information was missing from P2900R14 and found that the paper is self-contained for its target audience (CWG, LWG) while the companion paper serves the broader audience. *This architecture withstands opposition.*

Approbatio 11: Mixed-Mode Treatment (Section 3.5.11). The paper addresses the interaction between different evaluation semantics in different translation units. It explains why this is not an ODR violation (same token sequences, different generated code --- analogous to different optimization levels). It enumerates the conforming implementation strategies: inlining with local configuration, TU-local definitions, link-time ordering, and consistent-TU requirements. P3846R0 Concern 2 and Concern 4 provide detailed defense. The Advocatus filed a candidate charge that mixed-mode is confusing; the Dei dismissed it under Humanitas (no committee member who understands the ODR would be confused by the argument) and Confessio (the paper dedicates two pages to explaining the behavior). *This section withstands opposition.*

Objections

Objection 1: The Gap Between the Safety Narrative and the Normative Guarantees

Test failed: Ratio (logical soundness).

Quoted text:

Section 3.5.5, Selection of Semantics:

> *"The mechanism by which the evaluation semantic (ignore, observe, enforce, or quick-enforce) for any particular evaluation of a contract assertion is chosen is implementation-defined."*

Proposed wording, paragraph 3:

> *"Recommended practice: An implementation should provide the option to translate a program such that all evaluations of contract assertions use the ignore semantic as well as the option to translate a program such that all evaluations of contract assertions use the enforce semantic. By default, evaluations of contract assertions should use the enforce semantic."*

Section 3.1.1, Principle 1 (Prime Directive):

> *"The presence or evaluation of a contract assertion in a program should not alter the correctness of that program."*

Gravamen: The paper's design motivation is grounded in safety and correctness. Sixteen design principles, beginning with the Prime Directive, build a narrative in which contract assertions are instruments for identifying bugs and improving program reliability. The paper's sole normative protection ensuring that any contract is actually checked in practice --- the recommended practice for enforce-by-default --- carries no normative weight in the ISO standard. A conforming implementation that provides only the ignore semantic, or that defaults to ignore without user action, is fully conforming. The gap between the paper's safety-motivated design narrative and the normative guarantees available to users is wider than the paper's framing suggests.

This is not a charge that the design is wrong. The Advocatus examined the alternative --- mandating a specific default semantic --- and found it rejected by EWG at Tokyo (6--1--3--15--24 against enforce-only). The implementation-defined model serves the diversity of C++ platforms: embedded systems, game engines, safety-critical avionics, and student compilers all have different needs. The design is defensible.

The charge is narrower: the paper presents recommended practice as if it closes the gap between motivation and specification, but recommended practice is non-normative in ISO standards. A reader of P2900R14 who is not an ISO process expert may believe that "Recommended practice: ... should use the enforce semantic" provides a meaningful default guarantee. It does not. It provides guidance that implementations are free to ignore.

Motivatio:

Named adversary: Safety-focused national bodies --- Romania (RO-056), Finland (FI-071), Spain (ES-049, ES-050), France (FR-053, FR-054), and United States delegates (US-051, US-052) --- who filed NB comments requesting removal of contracts or removal of the ignore semantic. Also: the authors of P3608R0, who proposed deferring contracts to C++29 on the grounds of insufficient safety guarantees.

Named forum: NB comment resolution at plenary; EWG discussion during Sophia Antipolis.

Named damage: If this attack lands, the most likely outcome is not paper removal (contracts have survived plenary with strong consensus and have been retained through NB comment resolution at Croydon in March 2026). The more likely damage is a requirement to either (a) strengthen the recommended-practice language to something with more normative weight, (b) add a normative minimum requirement that conforming implementations shall support the enforce semantic, or (c) qualify the safety claims in Section 3.1 with explicit acknowledgment that checking depends on quality of implementation. Any of these outcomes is manageable. None requires redesign.

Hardening recommendation (defensio): Address this before the audience by explicitly bridging the gap. In presentation materials and in future revisions of the design rationale, state: "The standard specifies the mechanism. Quality of implementation provides the guarantee. Recommended practice guides implementations toward the safety default. This is the same model C++ uses for diagnostics, optimization, and error messages --- the standard defines what is conforming; the market determines what is useful." This framing converts a perceived weakness into a described design philosophy. Additionally, the postulator should be prepared to cite the implementations that already exist (GCC and Clang forks) as evidence that recommended practice IS being followed, even before the standard ships.

Notae Minores

The following observations were filed as candidate charges and relegated by the Advocatus Dei under the Dignitas challenge. They are editorial or cosmetic. The postulator reads them at their discretion.

1. **The keyword `contract_assert` is verbose.** At sixteen characters, it is among the longest keywords in C++. The paper acknowledges the choice was necessary to avoid collision with the `assert` macro (Section 3.2.2). Users writing assertion statements in tight loops or deeply nested code may find the length burdensome. This is an ergonomic observation, not a defect. The paper's rationale is sound: disambiguation from a function call requires a full keyword, and `assert` is taken.
2. **The paper's length challenges reviewability.** At 119 pages (4348 lines of text), P2900R14 is one of the longest papers in recent WG21 history. The design section alone spans 57 pages before the proposed wording begins. While the length is justified by the scope of the proposal, it creates a practical burden for reviewers who must understand both the design rationale and the formal wording. The companion paper P2899R1 adds another 100+ pages. The postulator has mitigated this through clear structure, a detailed table of contents, and cross-references between design and wording sections.
3. **The paper cites [P2899R1] for essential context seven or more times.** A reader of P2900R14 alone may be left without sufficient justification for certain design decisions, particularly the historical rationale for excluding features listed in Section 2.3. While the companion-paper architecture is sound (see Approbatio 10), the dependence on a separate document for motivation creates a risk that reviewers will evaluate the design section without the rationale. The postulator should consider whether a brief "rationale summary" subsection within P2900R14 would reduce this risk.

The Acta

This section records the audit trail of the examination for import into any future cause involving P2900 or the Contracts domain.

Phase I --- Citatio (Rules 1--3):

- The tribunal convened on April 27, 2026 to examine P2900R14, "Contracts for C++," dated February 13, 2025.
- Classification: Ask-paper. The paper proposes adoption of a Contracts feature into C++26.
- Posture: *Defensio*. The postulator seeks hardening of their own work.
- The *caput causae* (central thesis): C++ requires a standardized Contracts MVP consisting of three kinds of contract assertions (preconditions, postconditions, assertion statements) with four implementation-defined evaluation semantics and a replaceable violation handler, designed as a minimal but extensible foundation.
- Four readings performed. Ten factual articuli extracted. Eleven normative articuli extracted. Boundaries identified: the paper explicitly disclaims completeness, does not claim to replace `assert`, does not propose assertion levels, assume semantic, virtual function contracts, or invariants.

Phase II --- Inquisitio (Rules 4--6):

- *Public record search:* Web searches conducted for P2900R14 reception, committee discussion, NB comments, competing proposals, and stakeholder positions. Key findings:
  - P2900R14 adopted into C++26 working draft at Hagenberg (February 2025) with strong consensus.
  - Seven national body comments requested removal: ES-049, ES-050, US-051, US-052, FR-053, FR-054, FI-071.
  - Additional NB comments requested redesign of specific features: RO-056 (remove ignore), ES-055 and CZ-058 (redesign constification).
  - Three competing proposals identified: P4043R0 (readiness question), P4009R0 (major redesign), P3608R0 (defer to C++29).
  - P4020R0 by Andrzej Krzemienski (co-author of P2900) raised concerns about deployment experience, published February 2026.
  - P3846R0 assembled 22 co-authors to respond to all 17 recurring concerns, published October 2025.
  - Contracts retained at EWG Croydon (March 2026) after NB comment resolution.
- *Indexed archive search:* No MCP indexes queried (not configured for this cause).
- *Local workspace search:* No prior advocatus causes found against P2900 or in the Contracts domain.
- *Stakeholder dossier:* Seven national bodies with published opposition positions. Three counter-proposal authors. Twenty-two named defenders in P3846R0. Implementation experience from GCC and Clang forks (P3460R0).
- *Acta Priora:* No prior causes. This is a first hearing.
- *Citation verification:* Seven citations in P2900R14's bibliography. Resolution attempted for each. See Tabula Fontium below.

Phase III --- Interrogatio (Rules 7--9):

- **Skipped.** The postulator was not available for deposition. The Advocatus proceeded on the positio and public record only. Assumptions were classified as *Acta* (verified from public record) where possible. Unverified assumptions were noted and treated conservatively --- no charge was filed on unverified ground.

Phase IV --- Examen (Rules 10--15):

- Twenty-one articuli examined (10 factual, 11 normative).
- Three tests applied per articulus: Veritas, Ratio, Auctoritas.

- All factual articuli passed all three tests.
- Approximately fifteen candidate charges filed across normative articuli.
- Advocatus Dei cross-examination results:
  - Twelve charges withdrawn under **Confessio** --- the paper explicitly concedes the point under attack. Key confessions: side-effect elision consequences (S3.5.8), shallow const limitation (S3.4.2), virtual function exclusion (S2.3), implementation-defined semantics (S3.5.5), coroutine postcondition limitations (S3.5.2), throwing handler behavior (S3.6.6), ODR implications of side-effect elision in constant evaluation (S3.5.12).
  - One charge withdrawn under **Articulus** --- the paper does not claim what the objection attacks (interaction with future language features).
  - One charge withdrawn under **Humanitas** --- no real human opponent would raise the argument (exhaustive analysis of all possible predicate expressions).
  - Three observations relegated under **Dignitas** to Notae Minores.
  - **One charge survived all six challenges** and was filed as Objection 1 (the safety-narrative gap).
- Approbatio granted to eleven sections or claims where the Dei prevailed.
- Cause weighed (Rule 15): The central thesis survives the examination. The paper achieves what it sets out to do. The one surviving objection concerns framing, not substance. The paper's foundation is sound. Three minor cracks in the periphery do not undermine the structure.

Tabula Fontium

#	Citation	Link	Resolution	Status
1	[CWG2841] Tom Honermann, "When do const objects start being const?"	`wg21.link/cwg2841`	Resolves via wg21.link	**Resolved**
2	[N4993] Thomas Koeppe, "Working Draft, Programming Languages -- C++"	`wg21.link/N4993`	Resolves via wg21.link	**Resolved**
3	[P1494R5] S. Davis Herring, "Partial program correctness"	`wg21.link/P1494R5`	Resolves via wg21.link	**Resolved**
4	[P2053R1] Rostislav Khlebnikov and John Lakos, "Defensive Checks Versus Input Validation"	`wg21.link/P2053R1`	Resolves via wg21.link	**Resolved**
5	[P2695R0] Timur Doumler and John Spicer, "A proposed plan for contracts in C++"	`wg21.link/P2695R0`	Resolves via wg21.link	**Resolved**
6	[P2899R1] Doumler, Berne, Krzemienski, Khlebnikov, "Contracts for C++ -- Rationale"	`wg21.link/P2899R1`	Resolves via wg21.link	**Resolved**
7	[dcl.fct.def.coroutine]/5, /13	N/A (standard cross-ref)	Internal reference to C++ working draft	**N/A**
8	[class.temporary]/3	N/A (standard cross-ref)	Internal reference to C++ working draft	**N/A**
9	[expr.const]	N/A (standard cross-ref)	Internal reference to C++ working draft	**N/A**
10	[expr.await]/5.1.3	N/A (standard cross-ref)	Internal reference to C++ working draft	**N/A**

All six external citations resolve. No quote-mismatch detected. No unresolved third-party references. Standard-internal cross-references are not independently verifiable but are consistent with the working draft section numbering.

Closing Seal

Cum obiectionibus.

The cause proceeds with one objection. The gap between the paper's safety-motivated design narrative and the non-normative character of the recommended-practice default should be addressed before EWG --- not because the design is wrong, but because the framing invites an attack the postulator can preempt.

The tribunal is adjourned. The Advocatus returns the brief.