Contracts for C++: Naming the "Louis Semantic"

Timur Doumler (papers@timur.audio)

Document #: P3226R0
Date: 2024-04-12
Project: Programming Language C++
Audience: SG21

Abstract

At the March 2023 Tokyo meeting, SG21 adopted [P3191R0] for the Contracts MVP, which introduces a fourth contract evaluation semantic, informally known as the "Louis semantic". However, we do not yet have a name for this semantic. In this paper, we collect all names that have been suggested on the SG21 reflector so far, along with the different design criteria. We present the results of an electronic poll offering insights into which of these design criteria SG21 considers most important, conduct an analysis of which suggested names best satisfy these design criteria, and identify one name for the "Louis semantic" that best satisfies all relevant criteria.

1 Introduction

The current Contracts MVP paper [P2900R6] proposes three *evaluation semantics* for contract assertions:

— `ignore` – do not evaluate the predicate;

— `observe` – evaluate the predicate; if a contract violation has been detected, invoke the contract-violation handler; when the contract-violation handler returns, continue execution of the program;

— `enforce` – evaluate the predicate; if a contract violation has been detected, invoke the contract-violation handler; when the contract-violation handler returns, terminate the program in an implementation-defined fashion.

At the March 2023 Tokyo meeting, SG21 adopted [P3191R0] for the Contracts MVP, which introduces a fourth evaluation semantic:

— evaluate the predicate; if there is a contract violation has been detected, do not invoke the contract-violation handler; instead, immediately terminate the program in an implementation-defined fashion.

However, the adopted paper did not propose a name for the new, fourth evaluation semantic. It has been informally called the "Louis semantic" after the first author of [P3191R0], Louis Dionne. In order to ship the Contracts MVP, we need to agree on a proper name.

While this name will not be directly exposed in the user-facing API proposed in [P2900R6], as the enum `evaluation_semantic` only contains values for the evaluation semantics that can call the contract-violation handler, the name will be reflected in the wording that specifies the behaviour of contract assertions, and will be used to teach usage of the Contracts facility to users of C++. Therefore, we need to choose the new name carefully.

The need for a name for the fourth evaluation semantic sparked a lively discussion on the SG21 reflector. In the course of the discussion, dozens of solutions have been proposed by various people. These solutions can be categorised into three families:

A. Leave the names for the existing three semantics, `ignore`, `observe`, and `enforce`, unchanged; add a name for the fourth semantic that fits the existing naming pattern. Many such names have been suggested: `ensure`, `enforce_fast`, `trap`, etc.

B. Choose a new set of names for all four evaluation semantics that describes the behaviour of each semantic in terms of their constituent operations (call the contract-violation handler yes/no, terminate the program yes/no). For example, we could reassign the name `enforce` to mean the new fourth semantic, and rename the old `enforce` semantic to `observe_and_enforce` to express that it is a combination of `observe` (call the contract-violation handler) and `enforce` (terminate the program). In addition, we could swap the verbs `observe` and/or `enforce` in this set of names for more descriptive ones such as `call_handler` and `terminate`, respectively.

C. Do not codify names for the four evaluation semantics at all; instead, name the possible orthogonal operations (call the contract-violation handler yes/no, terminate the program yes/no) that constitute these evaluation semantics.

These three families of solutions have different motivating design goals and concerns. The goal of this paper is to understand which of these design goals SG21 considers most important and relevant for the Contracts MVP, identify the name that best satisfies these design goals and has the best potential for achieving consensus in SG21, and then propose this name for adoption into the Contracts MVP.

2 Methodology

We collected all design goals, requirements, or principles that have been mentioned on the SG21 reflector as motivation for choosing particular names. We then conducted an online survey, in which we asked participants to rank each of these design goals into one of the following three priority levels: **Must Have** (will vote against any solution that does not fully satisfy this requirement), **Important** (not a must-have, but should be prioritised against other design goals), and **Not Important**.

14 members of WG21 participated in the survey, which we consider reasonably representative. In comparison, 13 members of WG21 participated in the earlier [P2885R3] survey that was instrumental in getting consensus on a Contracts syntax.

We deliberately chose a simplified version of the process described in [P3004R0] and previously used in the [P2885R3] syntax survey: we used only three priority levels instead of five, and we did not ask whether the given criterion is objective or subjective. As we will see, this is actually sufficient as we can narrow the suggested solutions down to a sufficiently small set of candidates by considering just the Must Have criteria, so finer-grained priority levels are not needed to achieve the goal of this paper.

We asked participants to rank 38 different criteria (which prompted some criticism for being too many). Some of these criteria overlap. Others are contradictory. Some are relatively vague or generic principles, while others are very specific requests. All this is intentional as we want to fairly represent the full breadth of opinions expressed on the SG21 reflector.

The full results of the survey, including the full wording of the criteria, are summarised in the Appendix; below, we use abbreviated descriptions for the criteria. We list the relevant criteria with these abbreviated descriptions, their priority rank (1 — 38, with 1 being the one with the highest priority), and the voting result (Must Have — Important — Not Important). Priority is ranked by number of votes for Must Have; if this number is equal, by number of votes for Important.

As we will see in the following analysis, no solution exists that *perfectly* satisfies all criteria that someone considers a Must Have. The goal of our analysis is to find a solution that at least *partially* satisfies all such criteria, i.e. one that does not outright violate any of them. In cases where we need to make tradeoffs between different criteria, we will prefer solutions that more fully satisfy the higher-ranked criteria, i.e. those that more participants of the survey considered a Must Have.

3 Choosing the best family of solutions

Using just the top 10 criteria, which all have 4 or more Must Have votes, we can already exclude two of three families of solutions A — C listed in Section 1 because they are incapable of satisfying several of these criteria:

Rank	Short description	MH	I	NI	Family A	Family B	Family C
1	No semantic name should be an equally good or better fit for a different semantic	6	5	3	yes	yes	yes
2	Avoid the term “safe”	6	5	3	yes	yes	yes
3	Easy to understand by novices	5	7	2	yes	yes	yes
4	Minimise risk of MVP missing C++26	5	5	4	?	?	?
5	Allow adding `assume` semantic post-MVP	4	9	1	yes	yes	yes
6	Names should not be confused	4	7	3	yes	yes	yes
7	Describe what the semantic conceptually does with a contract assertion	4	6	4	yes	no	no
8	Easy to use in compiler flags such as `-fcontracts=<semantic>`	4	6	4	yes	yes	no
9	Do not change shape of P2900R6 library API	4	4	6	yes	yes	no
10	Do not rename `ignore`, `observe`, `enforce`	4	1	9	yes	no	no

We do not have a metric to evaluate how well any given solution satisfies criterion 4 (Minimise risk of MVP missing C++26), so we will ignore it for now. Looking at the other nine top 10 criteria, we see that only Family A can provide a solution that sufficiently satisfies them all.

Family B is designed to satisfy criterion 12 (describing what each semantic does when there is a contract violation, in terms of its constituent operations: calling the contract-violation handler and/or terminating the program) instead of the higher-ranked criterion 7 (describe what the semantic conceptually does with a contract assertion — ignoring it, enforcing it, etc.). In addition, Family B by design fails criterion 10 (do not rename `ignore`, `observe`, `enforce`).

Family C also fails the same criteria 7 and 10. In addition, Family C fails criterion 8 (easy to use in compiler flags such as `-fcontracts=<semantic>`) because it does not provide explicit names for the evaluation semantics, and criterion 8 (do not change the shape of the current library API) because it does away with these explicit names.

We therefore will consider only solutions from Family A for the remainder of this paper.

4 Narrowing down the candidate solutions

We now know that we want to restrict ourselves to solutions that leave the names for the existing three semantics, `ignore`, `observe`, and `enforce`, unchanged, and add a name for the fourth semantic that fits the existing naming pattern.

The following 25 names for the fourth semantic have been proposed on the SG21 reflector at the time of writing:

`abend`	`exit`	`panic`
`abort`	`fail`	`quick_enforce`
`cease`	`failfast`	`quit`
`crash`	`failsafe`	`raw_enforce`
`die`	`fast_enforce`	`stop`
`end`	`halt`	`terminate`
`enforce_fast`	`hard_enforce`	`trap`
`enforce_immediately`	`kill`	
`ensure`	`nohandler_enforce`	

We can narrow down this list by going through all criteria that have multiple Must Have votes, starting with the highest-ranked one, and exclude any names that fail the criterion.

`ensure` fails criterion 1 (no semantic name should be an equally good or better fit for a different semantic) because it is not sufficiently differentiable from `enforce`.

`failsafe` fails criterion 2 (avoid the term "safe").

`abend` arguably fails criterion 3 (easy to understand by novices) as it is a technical term (an abbreviation of "abnormal end") that is not an English word and is not very common nowadays. Similarly, `raw_enforce` and `hard_enforce` arguably fail this criterion because it is not obvious at all what "raw" or "hard" means in this context.

The next criterion that is not satisfied by some of the names is criterion 7 (describe what the semantic conceptually does with a contract assertion — ignoring it, enforcing it, etc.). This criterion is not satisfied by any of the names that merely express that the program is terminating (`abort`, `cease`, `crash`, etc.). Most of these names also fail criterions 14 and 16 (with 3 and 2 Must Have votes, respectively) which say that the name should avoid the names of specific termination functions that exists in the C++ standard library today (`terminate`, `abort`, `exit`) as well as the names of functions on existing popular C++ platforms (POSIX, Windows, etc.) and CPU instructions on popular architectures that terminate the program in a specific way (`halt`, `quit`, `trap`, `kill`) as the MVP does not prescribe any particular way in which the program should be terminated.

The following 5 names are thus the only ones that satisfy all top 10 criteria (criteria with 4 or more Must Have votes) at least to some degree:

`enforce_fast`
`enforce_immediately`
`fast_enforce`
`nohandler_enforce`
`quick_enforce`

All of these describe that the semantic is conceptually *enforced* (thereby satisfying criterion 7) by terminating the program, and add a qualifier to the word `enforce`.

Criterion 8 (easy to use in compiler flags) is satisfied better by the shorter names, as they are easier to type, but all names are in principle usable in compiler flags.

If we go further down the list of criteria and consider criteria that have fewer than 4 Must Have votes, we find a few more criteria that not all names satisfy equally well.

Criterion 11 (easy to use in post-MVP labels; 3 Must Have votes) is better satisfied by shorter names, similarly to criterion 8.

Criterion 12 (describing what each semantic does when there is a contract violation, in terms of its constituent operations: calling the contract-violation handler and/or terminating the program; 3 Must Have votes) is well-satisfied by `nohandler_enforce`, but only partially satisfied by the other remaining names, as they do not spell out that the contract-violation handler is being called (but they at least hint at this property by using the word `enforce` with an additional qualifier).

Criteria 15 (2 Must Have votes), 17 (2 Must Have votes), and 22 (1 Must Have vote) are all very similar to criterion 12.

Criterion 29 (all four names should have a similar length, 1 Must Have vote) is less well satisfied by the longer names.

Criteria 30 and 31 (being able to consistently apply the names of all semantics as a part of speech to the same subject and object) are satisfied by `enforce_fast`, `enforce_immediately`, and `fast_enforce`, because these consist of a verb plus an adverb and are usable in the same position in the sentence as the plain verb `enforce`. These criteria are satisfied somewhat less well by `quick_enforce`: `quickly_enforce` would be more grammatically correct here, but that was not on the list of proposed names. However, the grammar still works if we consider "quick-enforce" a compound verb similar to many other such verbs in technical language: "double-click", "force-quit", "compare-exchange" etc. These criteria are not satisfied by `nohandler_enforce`, as this does not make a grammatically acceptable English sentence no matter how much you squint.

Finally, Criterion 33 (each name should start with a different letter; 1 Must Have vote) is only satisfied by the names that add a prefix before `enforce`: `fast_enforce`, `nohandler_enforce`, and `quick_enforce`, and not by those who add a suffix.

There are a handful of remaining criteria at the end which none of the survey participants considers a Must Have, so we do not consider them particularly relevant for our choice of names.

If we consolidate the overlapping relevant criteria into common properties of the five remaining candidate names, we get the following table (ordered by priority in descending order):

Property	`enforce_fast`	`enforce_immediately`	`fast_enforce`	`nohandler_enforce`	`quick_enforce`
Name length in characters (existing semantics have $6-8$)	12	19	12	17	13
Describing behaviour	partially	partially	partially	yes	partially
Consistent as part of speech	yes	yes	yes	no	partially
Different letter for each name	no	no	yes	yes	yes

We therefore conclude that the only candidate names that do not outright violate any of the criteria categorised as Must Have by at least one survey participant are `fast_enforce` and `quick_enforce`.

On the one hand, `fast_enforce` is one character shorter than `quick_enforce`, therefore it satisfies the criteria that concern the length of the name slightly better, and is more grammatically correct when used as a verb in an English sentence.

On the other hand, `quick_enforce` has a significant advantage over `fast_enforce` that is not captured in the online survey criteria: better consistency with existing practice. The C++ standard library already contains a facility that terminates the program and comes in two flavours, one that calls a user-defined callback and one that does not: `std::exit` and `std::quick_exit`. The difference between the two is that the former calls functions registered with `std::atexit` before the program is terminated, while the latter does not.

It seems very compelling to call the two terminating contract evaluation semantics `enforce` and `quick_enforce`, where the former calls the contract-violation handler before the program is terminated, while the latter does not, thus following the same naming scheme as `std::exit` and `std::quick_exit`. We believe that this neat property of the name `quick_enforce` outweighs the minor disadvantages that this name has over `fast_enforce`. We therefore conclude that `quick_enforce` is the best name for the "Louis semantic" out of the 25 names that were suggested on the SG21 reflector.

5 Proposal

Adopt `quick_enforce` as the name for the new, fourth evaluation semantic adopted into the Contracts MVP via [P3191R0], until now informally known as the "Louis semantic".

Acknowledgements

We would like to thank all WG21 members who suggested names for the "Louis semantic", suggested design criteria for these names, and participated in the online survey.

References

[P2885R3] Timur Doumler, Gašper Ažman, Joshua Berne, Andrzej Krzemieński, Ville Voutilainen, and Tom Honermann. Requirements for a Contracts syntax. https://wg21.link/p2885r3, 2023-10-02.

[P2900R6] Joshua Berne, Timur Doumler, and Andrzej Krzemieński. Contracts for C++. https://wg21.link/p2900r6, 2024-02-29.

[P3004R0] John Lakos, Harold Bott, Mungo Gill, Lori Hughes, Alisdair Meredith, Bill Chapman, Mike Giroux, and Oleg Subbotin. Principled Design for WG21. https://wg21.link/p3004r0, 2024-02-13.

[P3191R0] Louis Dionne, Yeoul Na, and Konstantin Varlamov. Feedback on the scalability of contract violation handlers in P2900. https://wg21.link/p3191r0, 2023-03-19.

Appendix: Electronic poll results

Rank	Full description	MH	I	NI
1	No evaluation semantic should have a name that could be an equally good or better fit for a different semantic.	6	5	3
2	The names for all four evaluation semantics, including the new Louis semantic, should avoid the term "safe" because that term means too many different things to different people.	6	5	3
3	The names for all four evaluation semantics, including the new Louis semantic, should be easy to understand by novices.	5	7	2
4	We should not make changes to P2900R6 that could increase the probability of the Contracts MVP missing C++26.	5	5	4
5	The names for all four evaluation semantics, including the new Louis semantic, should be chosen in a way that post-MVP, we can add an assume semantic without compromising consistency.	4	9	1
6	The names for all four evaluation semantics, including the new Louis semantic, should be sufficiently descriptive so that they are never confused.	4	7	3
7	All four evaluation semantics, including the new Louis semantic, should have a name descriptive of what the semantic conceptually chooses to do with a contract assertion when evaluating it (ignoring it, enforcing it, etc.)	4	6	4
8	All four evaluation semantics, including the new Louis semantic, should have a name that is easy to use in compiler flags such as -fcontracts=<semantic>.	4	6	4
9	Irrespective of names, we should not change the design of the contract-violation handling API in P2900R6: a contract violation type with five member functions having the currently specified behaviour, three enums having the currently specified meaning, and a free function to invoke the functionality of the default contract-violation handler.	4	4	6
10	We should not rename the three pre-existing evaluation semantics in P2900R6: ignore, observe, enforce.	4	1	9
11	All four evaluation semantics, including the new Louis semantic, should have a name that is easy to use in post-MVP labels on contract assertions.	3	9	2
12	All four evaluation semantics, including the new Louis semantic, should have a name descriptive of what the semantic does (its observable behaviour) when there is a contract violation (call the contract-violation handler yes/no, terminate the program yes/no).	3	7	4
13	The names for the four evaluation semantics should be descriptive of the fact that the P2900 ignore semantic is not allowed to evaluate the predicate, whereas the other three semantics are.	3	4	7
14	The names for the two semantics that are specified to terminate the program in an implementation-defined fashion (the P2900R6 enforce semantic and the new Louis semantic) should avoid the names of specific termination functions that exists in the C++ standard library today, such as terminate, abort, exit.	3	4	7
15	The name for the new Louis semantic should be descriptive of the fact that on contract violation, the program is terminated directly, without calling the contract-violation handler.	2	9	3

16	The names for the two semantics that are specified to terminate the program in an implementation-defined fashion (the P2900R6 enforce semantic and the new Louis semantic) should avoid the names of concrete functions or CPU instructions used on existing popular C++ platforms (POSIX, Windows, etc.) to terminate the program in a specific way, such as halt, quit, trap, kill.	2	7	5
17	Given that the four evaluation semantics constitute the four possible combinations of two orthogonal operations (call the contract-violation handler yes/no, terminate the program yes/no), each semantic should have a name that makes it obvious to the user which of the four possible combinations it represents.	2	5	7
18	The name for the P2900R6 enforce semantic should be descriptive of the fact that on contract violation, it calls the contract-violation handler (whereas the new Louis semantic does not).	2	5	7
19	The design should allow for vendor-specific extensions that specify evaluation semantics different from the four proposed ones, and allow the user to identify these additional semantics inside the contract-violation handler.	2	4	8
20	All four evaluation semantics, including the new Louis semantic, should have a name that is suggestive of the primary intended use case of the given evaluation semantic.	2	4	8
21	The name for the P2900R6 enforce semantic should not contradict the fact that on contract violation, it may be possible to avoid program termination by throwing an exception from the contract-violation handler (whereas with the new Louis semantic, this is not possible).	2	3	9
22	The name for the new Louis semantic should reflect that it does nothing else beyond terminating the program on contract violation.	1	9	4
23	All four evaluation semantics, including the new Louis semantic, should have a name that does not use any rare, uncommon English words that many non-native speakers will be unfamiliar with.	1	7	6
24	Whether the chosen evaluation semantic will terminate the program should be queryable directly in the contract-violation handler via a named boolean function rather than indirectly through the names of the possible evaluation semantics.	1	4	9
25	Each of the four evaluation semantics, including the new Louis semantic, should have a name descriptive of that semantic, without having to know the names of the other semantics for understanding.	1	4	9
26	All four evaluation semantics, including the new Louis semantic, should have a name that is a verb.	1	4	9
27	The name for the P2900R6 enforce semantic should be descriptive of the fact that it combines the behaviour of the observe semantic (that the contract-violation handler is called) with that of the new Louis semantic (that the program will be terminated).	1	3	10
28	All four evaluation semantics, including the new Louis semantic, should have a name consisting of a single word.	1	3	10
29	The names for all four evaluation semantics, including the new Louis semantic, should have a similar length (amount of characters).	1	3	10

30	All four evaluation semantics, including the new Louis semantic, should have a name x that consistently applies as a part of speech to the same subject and object as follows: "this evaluation semantic xs the contract assertion".	1	3	10
31	All four evaluation semantics, including the new Louis semantic, should have a name x that consistently applies as a part of speech to the same subject and object as follows: "this evaluation semantic xs the truth of the contract predicate".	1	3	10
32	The name for the new Louis semantic should not imply that no information about the contract violation is available (while this semantic does not call the contract-violation handler at runtime, such information may still be made available through other means, for example via a symbolicated crash log).	1	2	11
33	Each of the four evaluation semantics, including the new Louis semantic, should have a name that starts with a different letter.	1	1	12
34	The names for all four evaluation semantics, including the new Louis semantic, should consist of English words that can be found in a standard dictionary such as the Oxford or Merriam-Webster English Dictionary.	0	6	8
35	Rather than naming each possible evaluation semantic, we should name the possible orthogonal behaviours (call the contract-violation handler yes/no, terminate the program yes/no), so that if we introduce more such orthogonal behaviours in the future, we can more easily integrate them into the design.	0	5	9
36	The name for the new Louis semantic should reflect that it is useful for achieving a smaller binary size compared to the P2900R6 enforce semantic.	0	4	10
37	The name for the new Louis semantic should reflect that it is useful for hardening a binary against security vulnerabilities.	0	1	13
38	Each of the four evaluation semantics, including the new Louis semantic, should have a name that starts with a vowel.	0	1	13