

Doc. No. P3970R0

Date: 2026-01-16

Audience: EWG, SG12, SG20, SG23

Author: David Vandevoorde

Jeff Garland

Paul E. McKenney

Roger Orr

Bjarne Stroustrup

Michael Wong

Reply to: daveed@vandevoorde.com

Profiles and Safety: a call to action

There are massive, well-founded demands for improved safety and simpler use of C++. SG23 and EWG have repeatedly (by massive votes) pointed to “Profiles” as the direction for addressing these urgent needs (e.g., [Safety Profiles: Type-and-resource Safe programming in ISO Standard C++](#) - the 2023 introduction of Profiles to EWG – and [What are profiles?](#) - a brief summary of the aims of profiles with plenty of references). However, we still face a stream of uncoordinated proposals to address problems from different perspectives, often not even mentioning Profiles. The Profiles intellectual framework (based on [the subset-of-superset strategy](#)) has been developed over several years to cover a wide range of requests involving a variety of notions of safety and style involving combinations of compile-time and run-time techniques. The way to make progress is to build on the proposed Profiles framework ([C++ Profiles: The Framework](#)) and start experimenting with specific profiles within that framework.

For that to happen, we need the framework available and accessible on one or more C++ implementations, ideally on all major implementations. Otherwise, using different initial/experimental profiles will require too much boilerplate code and different interfaces to different tool chains (e.g., in-code annotations, compiler options, and build-system settings). That will impede portability and possibly introduce incompatibilities that will be hard to root out in the future. Currently, there is – to the best of our knowledge – a [specification](#) and an experimental implementation is being conducted in a major C++ compiler. Also, an implementation guide is being written. We encourage implementers to coordinate and if possible collaborate to avoid inessential incompatibilities.

Beyond the framework implementation(s), we need to design and precisely specify a few initial profiles. We need a reasonable common style of Profile descriptions. That – and bringing the older documents in line with the C++26 specification – is a necessary first step on the way to standardization. A start is made in [A framework for Profiles development](#). We suggest the following as plausible initial profiles:

- **Initialization:** no guarantees are possible without guaranteed initialization of all objects before their first use. The simplest implementation of this idea is to require explicit or

implicit initialization (e.g., default as for `std::string`) of every object. This points to the need for two levels of description of each profile:

- A brief statement of the guarantee offered, e.g., “every object is initialized.”
- An implementation guide that lists the places where an implementation (library of compiler) needs to take an action. [A Safety Profile Verifying Initialization](#) is an effort in that direction.
- **Ranges**: we have hardened libraries, but to provide a manageable guarantee, there must be a standard way of requesting their use and they must be complemented with compile-time prevention of subscripting raw pointers and similar uncheckable constructs (e.g., see [Dealing with pointer errors: Separating static and dynamic checking](#)).
- **Resources**: Every resource is held by an object with a destructor that releases it if/when it goes out of scope (RAII).
- **Education**: Enforce a restricted version of C++ aimed at keeping novices out of the many dark corners. [The C++ Core Guidelines](#) could be an inspiration, and the education study group (SG20) is interested. [Programming -- Principles and Practice Using C++](#) and [A Tour of C++](#) both follow such an approach.

The key profiles to get in place to address the most frequent “unsafety complaints” are

- **Invalidation**: No access through dangling pointers.
- **Arithmetic**: No implicit narrowing conversions. No overflow or underflow.

This has been described ([Type-and-resource safety in modern C++](#)) and prototyped ([Lifetime safety: Preventing common dangling](#)) but requires precise specification and serious static analysis so it is not ideal as part of initial implementation experiments.

After that, more profiles should be defined and implemented, notably a concurrency profile. Please note that not every profile should be standardized and that not every profile will be safety related.

The ideas outlined here need solid engineering to become reality: we encourage organizations with such expertise to apply it and organizations with money to finance such work. In particular, we encourage work on open source and on open implementation guidance.