P3589R0 2025-01-13 Reply-To: gdr@microsoft.com
Audience: EWG

C++ Profiles: The Framework

Gabriel Dos Reis
Microsoft

This document offers an overview of the design and rationale for the "C++ Profiles" framework (Stroustrup, 2024) as discussed in previous papers and presentations (Stroustrup, 2024) (Stroustrup & Dos Reis, 2023) (Stroustrup, 2023) (Stroustrup & Dos Reis, 2022) (Stroustrup, 2024). The framework is independent of any specific "profile" (e.g. memory safety profile, type safety profile, style profiles, etc.). This is *not* a competing proposal with respect to Herb Sutter's proposal (Sutter, 2025). Rather, the aim here is to "factor out" the profile framework protocol from considerations and discussions of any specific profile in order to help the community focus attention on the right sets of concerns in the appropriate contexts. Questions regarding the Profiles *framework* should, in general, be considered independently of a given specific profile. Conversely, questions regarding a given profile should be considered independently of the framework. Of course, there will arise questions regarding interactions between profiles or how a given profile should work given the general framework. The aim here is to help structure conversations around profiles in order to make forward progress.

1 DESIGN

At its core, the idea of "profiles" is to allow a programmer to state that they desire their program to abide by the conventional ISO C++ rules and *additional rules*. A set of such additional rules, called *profile*, **are not to change the meaning (i.e. set of permitted behaviors) of a well-formed program with no undefined behavior**. To address the contemporary challenge of memory safety concern, we need some standard profiles related to memory safety, guaranteed to be available in all C++ implementations. Herb Sutter's proposal (Sutter, 2025) is focused on such profiles, to plug into the general C++ profiles framework.

This document is not to rehash past conversations; it is not a new proposal; rather, it is to provide an overview and rationale for the profiles framework. As stated in (Stroustrup, 2024), without a common framework, the options from different suppliers addressing common problems (e.g., range errors) will be significantly different, and code relying on one cannot be relied on to work on another or to offer the same guarantees. This serious problem can be addressed by the options implementing specific Profiles, just as if they had been specified in-code.

A companion paper looks into the practical matters of implementation strategies.

1.1 AIMS

The Profiles framework aims to

1. Provide syntactic support for a program (fragment) to express the set of guarantees it enjoys, i.e. set of profiles enforced by the C++ translator. Each profile is formulated to

ensure the program is free of a certain class of problems (e.g. memory safety, resource-leak free) (Stroustrup & Dos Reis, 2022)
2. Support an open-ended set of profiles, some standard, some implementation-defined, some provided by third parties, etc
3. Provide a mechanism for a library to express an interface along with guarantees that it offers
4. Provide a mechanism for a program to take dependencies on other components (modules, libraries, etc) and to request specific guarantees from those components

A profile may have an effect on the runtime behavior of a program such as enabling runtime instrumentation (e.g. bound checking in array indexing). However, its static semantic effects are as-if applied only at translation phase 8. In particular, it is not possible for a program to change the outcome of overload resolution or template instantiation, nor is it possible to "SFINAE out" failure of a program to satisfy a profile constraint.

2 Proposed Wording

Part of the proposed wording in this section is directly borrowed from Sutter's paper (Sutter, 2025).

2.1 Diagnosable Rule

Failure to satisfy a constraint dictated by a profile is a diagnosable rule. Accordingly, modify paragraph [intro.compliance.general]/1 as follows:

> The set of *diagnosable rules* consists of all syntactic, ~~and~~ semantic, and profile rules in this document except for those rules containing an explicit notation that "no diagnostic is required" or which are described as resulting in "undefined behavior". An implementation is permitted to provide additional profile rules provided they are active only under the appropriate implementation-defined profile.

2.2 Profile Enablement

A profile is enabled at the beginning of a translation as the first declaration. A profile is designated by a profile-designator defined as follows. Add these new productions to the list or productions in paragraph [decl.attr.grammar]/1

profile-designator:
    profile-name
    profile-name (*profile-argument-list*_opt)
profile-name:
    attribute-token
profile-argument-list:
    profile-argument
    profile-argument-list , *profile-argument*
profile-argument:
    identifier

identifier : *expression*

The identifier preceding the colon in a profile-argument is not looked up ([basic.lookup])

2.2.1 Syntax

Modify paragraph [decl.pre]/1 as follows:

empty-declaration:
attribute-specifier-seq_opt ;

Modify paragraph [decl.pre]/15 as follows

An *empty-declaration* has no effect. The optional *attribute-specifier-seq* appertains to the *empty-declaration*.

2.2.2 Static Semantics

A profile may have a dynamic semantics (e.g. requesting array bound checking) in addition to static semantics. The static semantics is conceptually applied after translation phase 7.

Add a new subsection titled "Profile attributes [decl.attr.profile]"

A profile attribute is an attribute where the *attribute-token* is `profiles::enforce` or `profiles::suppress`. A profile attribute whose *attribute-token* is `profiles::enforce` shall be applied only to *empty-declaration* and that empty-declaration shall be the first *declaration*, if any, in the *translation-unit*.

The *profile-name* in a *profile-designator* identifies a *profile*, which is a set of additional language restrictions applied after translation phase 7.

The *dominion* of a profile *P* is the sequence of tokens starting after a profile attribute whose *profile-name* designates P and extending to the end of the translation unit. The additional language restrictions enabled by the profile *P* apply only to its dominion, except to the entire region of a *declaration* or *statement* with a profile attribute whose *attribute-token* is `profiles::suppress` and profile-name designates *P*.

If a declaration *D* appears in the dominion of a profile *P1*, all other redeclarations of *D*, if any, shall appear in the dominion a profile *P2* and any such *P2* shall be compatible with *P1*. All standard profiles are compatible with each other.

2.2.3 Linking Expectations to Guarantees

A module can explicitly advertise the profiles (guarantees) its interface provides.

Add a new paragraph to subsection [module.interface]

> If an enforcement profile attribute appears in the *attribute-specifier-seq*, if any, of a primary module interface unit ([module.unit]), then all module units of that module are in the dominion of the nominated profile.

Add a new paragraph to subsection [module.import]

> If an application profile attribute appears in the optional *attribute-specifier-seq* of a *module-import-declaration* then the *module-import-declaration* shall designate a named module, and the nominated profile *P* shall be designated in the declaration of the module interface unit. The dominion of *P* is extended to the end of the translation unit containing the module-import-declaration.

3 REFERENCES

Stroustrup, B., 2023. *Concrete suggestions for initial Profiles*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2023/p3038r0.pdf

Stroustrup, B., 2024. *A framework for Profiles development*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2024/p3274r0.pdf

Stroustrup, B., 2024. *Profile invalidation - eliminating dangling pointers*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2024/p3446r0.pdf

Stroustrup, B., 2024. *Profiles syntax*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2024/p3447r0.pdf

Stroustrup, B. & Dos Reis, G., 2022. *Design Alternatives for Type-and-Resource Safe C++*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2022/p2687r0.pdf

Stroustrup, B. & Dos Reis, G., 2023. *Safety Profiles: Type-and-resource Safe Programming in ISO Standard C++*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2023/p2816r0.pdf

Sutter, H., 2025. *Core safety Profiles: Specification, adoptability, and impact*. [Online]
Available at: https://open-std.org/JTC1/SC22/WG21/docs/papers/2025/p3081r1.pdf