# Split `define_aggregate` from Reflection

Document #: P3569R0
Date: 2025-01-11
Project: ISO/IEC JTC1/SC22/WG21 14882: Programming Language — C++
Audience: Evolution Working Group
Author: Shafik Yaghmour <shafik.yaghmour@intel.com>, Aaron Ballman <aaron@aaronballman.com>, Erich Keane <ekeane@nvidia.com>, Corentin Jabot <corentin.jabot@gmail.com>, Vlad Serebrennikov <serebrennikov.vladislav@gmail.com>

## Introduction

Five of Clang's maintainers have been working closely with the authors of P2996 over the past four months on concerns that we've identified when deeply exploring the implementation effort for Reflection. We greatly appreciate the collaboration with the paper authors as we've come up with corner cases and design questions. However, we have concerns with one aspect of the proposal: `define_aggregate`. The Reflection proposal (https://wg21.link/p2996) is large in scope, touching almost every major core language section in the standard. The `define_aggregate` API is one of the most complicated aspects to nail down the semantics for and has undergone several major reworkings in the past few revisions of the paper, including many after the approval by EWG. At this stage of the C++26 process, we believe this part of the proposal carries significant risks not only to the Reflection proposal as a whole but also to other proposals targeting C++26 (CWG bandwidth being limited).

While `define_aggregate` is a useful feature of Reflection, the proposal remains extremely valuable to users even without `define_aggregate`. We believe that `define_aggregate` can be safely split from the rest of the proposal. This gives the committee and implementers time to fully reflect on solutions to design challenges that have been discovered with this aspect of the greater proposal. Leveraging the train model allows implementations to deploy the better-understood parts of Reflection while allowing further implementation experience with `define_aggregate`. It also mitigates the risk of introducing major breaking changes into the standard after C++26.

## Some History

Much of the Reflection proposal deals with interrogating the internal state of the compiler's AST. While specifying this is a large amount of work, it is relatively low risk. Interrogating the type system is something C++ has supported for a long time via type traits and implementations frequently provide interfaces for tapping into that functionality already, so this is a relatively well-understood and non-contentious part of the proposal. Conversely, `define_aggregate` allows creating aggregates during constant evaluation which bring many novel challenges for implementations and the standard.

One of the challenges that came up during the Wrocław meeting was that we need to understand and solve issues around complete class contexts. CWG was uncomfortable solving the complete class context issues piecemeal, but the combined set of complete class context issues is large and onerous. Trying to solve that problem within the C++26 timeframe, without leading to ABI-impacting DRs to resolve in future versions of the standard, is risky and likely not plausible. After the Wrocław meeting, the authors made significant design changes to the proposal introducing new scoping rules in an attempt to resolve issues around complete class context, but this introduces new, novel design mechanics at a late stage into what is already a significantly large proposal.

Another design issue that came up during the post-Wrocław CWG wording review was that the proposal for `define_agggregate` is trying to specify a novel form of reachability.The standard has the concept of reachability already for modules, but unfortunately it is underspecified for the needs of `define_aggregate`. Although there are ideas for how to specify the wording, that is another significant reworking of the design of the feature and needs to be evaluated for correctness not only for `define_aggregate` but also that it doesn't negatively impact the behavior of modules. Because the wording is not yet available, it's not clear whether this new approach is viable, nor whether it is acceptable to Evolution.

In contrast, the rest of the wording for Reflection so far has been uncontroversial within CWG. We don't foresee problems with the wording outside of `define_aggregate`, so we suggest to split it out of the proposal. We believe we can approve the rest of the wording for Reflection quickly, which leaves time to focus solely on `define_aggregate` without risking the entire Reflection proposal if we can't find a sound approach in time for C++26.