Freestanding Library: Partial Classes

Document #:	
Date:	2021-05-11
Project:	Programming Language C++
Audience:	Library Evolution Working Group Library Working Group
Reply-to:	Emil Meissner <e.meissner@seznam.cz> Ben Craig <>

Contents

1 Introduction

This proposal is part of a group of papers aimed at improving the state of Freestanding. It marks (parts of) `std::array`, `std::string_view`, `std::variant`, and `std::optional` (hereinafter referred to as "the added classes") as such. A future paper might add `std::bitset` (as was the original goal in [P2268R0])

2 Motivation and Scope

All of the added classes are fundamentally compatible with Freestanding, except for a few methods that throw (e.g. `array::at`). We explicitly `=delete` these undesirable methods.

The main driving factor for these additions the immense use of these types in practice.

2.1 Scope

We refine [freestanding.membership] by specifying the notion of partial classes, and accordinly specify the added classes as (partially) freestanding.

2.2 Implementation experience

2.2.1 The Existing Standard Library

We made a fork of libc++. `=delete`ing all the methods was trivial, except for some methods on `string_view` (which are implemented in terms of the `=deleted` `string_view::substring`). All test cases (except for the `=deleted` methods) passed after some rather minor adjustments (e.g. replacing `get<0>(v)` with `*get_if<0>(&v)`).

2.2.2 In Practice

Since we aren't changing the semantics of any of the classes (except `=delete`ing non-critical methods), it is fair to say that all of the (implementer *AND* user) experience gathered as part of Hosted applies the same to Freestanding.

Therefore, we shouldn't be asking whether the *design* works, rather, whether it's technically feasible. To which the answer is yes! For example, the [Embedded Template Library] offers direct mappings of the `std` types. Even in kernel level libraries, like Serenity's [AK] use these utilities.

3 Design decisions

3.1 [conventions] changes

The predecessor to this paper used `//freestanding, partial` to signal a class require be only partially implemented, in conjunction with `//freestanding, omit` signaling a declaration is not required to be present in freestanding.

This paper chooses to go the more explicit way, that is marking each required declaration inside a class as `//freestanding`. It simultaneously introduces the concept of having two different declarations with the same name on freestanding vs hosted.

4 Wording

4.1 Change in [conventions]

Add new paragraphs to [freestanding.membership]

5 A declaration may be specified differently for freestanding implementations, in this case a hosted implementation shall use the variant not marked as freestanding, whereas a freestanding implementation shall use the variant marked as freestanding.

[*Example:*

```
    constexpr reference at(size_type n) = delete; // freestanding
    constexpr reference at(size_type n);
```

On a freestanding implementation `at` is deleted, whereas on a hosted one `at` is just a normal function. -end example]

6 A partially freestanding class or partially freestanding class template is a type which has at least one member that is either declared freestanding, or that differs from hosted as described in (5). [*Note:* These classes might be referred to as "partial classes" outside the standard. -end note]

7 Partially freestanding classe's or partially freestanding class template's members follow the same convetions as file scope declarations in regards to being marked as freestanding.

[*Example:*

```
class A {                 // freestanding
  int a();                // freestanding
  int b();
  int b() = delete;       // freestanding
  using Result = int;     // freestanding
};
```

Here, class `A` is a partially freestanding class, whose member type `Result` and member function `a` are both required on freestanding, but whose member function `b` is deleted on freestanding. -end example]

4.2 Changes in [compliance]

Add new rows to Table 24:

5 References

[AK] Andreas Kling. Serenity OS AK Library.
https://github.com/SerenityOS/serenity/tree/master/AK

[Embedded Template Library] John Wellbelove. Embedded Template Library.
https://www.etlcpp.com/

[P2268R0] Ben Craig. 2020-12-10. Freestanding Roadmap.
https://wg21.link/p2268r0