Freestanding Library: Partial Classes

Document #:	P2407R1
Date:	2021-07-11
Project:	Programming Language C++
Audience:	Library Evolution Working Group

Reply-to:	Emil Meissner <e.meissner@seznam.cz> Ben Craig <ben dot craig at gmail dot com>

Contents

1 Changes from previous revisions

1.1 Changes from R0

— Add wording for feature test macros
— Mention monadic optional and `string_view::contains`

2 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` as such. A future paper might add `std::bitset` (as was the original goal in [P2268R0])

3 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 is the immense usefulness of these types in practice.

3.1 Scope

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

3.1.1 About <bitset>

As mentioned in the introduction, this paper does not deal with bitset. Bitset is unique in that a relatively big part of its interface depends on `std::basic_string`. We do not currently have a sound plan to make bitset work as nicely as we'd like to. This situation is made worse by a significant amount of bitset's member functions that throw.

3.2 Implementation experience

3.2.1 The Existing Standard Library

We've forked libc++, and `=deleted` all not freestanding methods. Except for some methods on `string_view` (which are implemented in terms of the deleted `string_view::substring`), this did not require any changes in the implementation. 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)`), confirming that all these types are usable without the deleted methods.

3.2.2 In Practice

Since we aren't changing the semantics of any of the classes (except deleted 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.

The only question is, whether these classes are compatible with freestanding. 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 a form of these utilities.

4 Design decisions

4.1 Deleting behavior

Our decision to delete methods we can't mark as freestanding was made to keep overload resolution the same on freestanding as hosted.

An additional benefit here is, that users of these classes, who might expect to use a throwing method, which was not provided by the implementation, will get a more meaningful error than the method simply missing. This also means we can keep options open for reintroducing the deleted functions into freestanding. (e.g. `operator<<(ostream, string_view)`, should `<ostream>` be added).

4.2 [conventions] changes

The predecessor to this paper used `//freestanding, partial` to mean a class (template) is only required to be partially implemented, in conjunction with `//freestanding, omit` meaning a declaration is not in freestanding.

In this paper, we keep marking not fully freestanding classes templates as `//freestanding, partial`, requiring all members of such a class template to be individually marked as freestanding, or not. This is done to keep things explicit. We also introduce `//freestanding, delete`, to mean a declaration shall be deleted on freestanding.

4.3 On `std::visit`

In this paper, we mark `std::visit` as freestanding, even though it is theoretically throwing. However, the conditions for `std::visit` to throw are as follows:

> It is possible for a variant to hold no value if an exception is thrown during a type-changing assignment or emplacement.

This means a variant will only throw on visit if a user type throws (library types don't throw on freestanding). In this case, `std::visit` throwing isn't a problem, since the user's code is already using, and (hopefully) handling exceptions.

This however has the unfortunate side-effect that we need to keep `bad_variant_access` freestanding.

4.4 Notes on variant and value categories

By getting rid of `std::get`, we force users to use `std::get_if`. Since `std::get_if` returns a pointer, one can only access the value of a variant by dereferencing said pointer, obtaining an lvalue, discarding the value category of the held object. This is unlikely to have an impact on application code, but might impact highly generic library code.

5 Justification for deletions

Every deleted method is throwing. We omit `string_view`'s associated `operator<<` since we don't add `basic_ostream`.

6 Monadic optional and string_view::contains

Since this paper was first published, `std::string_view` got a new `contains` member function, and `std::optional` got `transform`, `and_then`, and `or_else`. All these functions are not throwing, and there are no other problems regarding freestanding. We therefore opt for them being marked as freestanding.

7 Wording

This paper's wording is based on the current working draft, [N4878], and it assumes that the wording in [P1642R5] and [P2338R0] has been applied.

7.1 Change in [conventions]

Add new paragraphs to [freestanding.membership]

5 A *freestanding member* is a member declaration of a freestanding class template that is implemented in freestanding implementations.

6 A *partially freestanding class template* is a freestanding class template, where at least one, but not all members are freestanding members. In the associated header synopsis for such a class template, the class template's declaration is followed with a comment that includes *freestanding* and *partial*.

[*Example:*

```
template<class T, size_t N> struct array; //freestanding, partial
```

-end example]

7 Each freestanding member in the synopsis of a partially freestanding class template is followed by a comment including *freestanding*.

8 *Deleted freestanding members* are member functions of a partially freestanding class template that are designated as such. Deleted freestanding members are not freestanding members. In the partially freestanding class template's synopsis, deleted freestanding members are followed with a comment that includes *freestanding* and *delete*. Deleted freestanding members shall either meet the requirements of a hosted implementation, or be deleted.

[*Example:*

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

-end example]

7.2 Changes in [compliance]

Add new rows to Table 24:

	Subclause	**Header(s)**
[...]	[...]	[...]
?.? [optional]	Optional Objects	<optional>
?.? [variant]	Variants	<variant>
?.? [string.view]	String view classes	<string_view>
?.? [array]	Sequence containers	<array>
[...]	[...]	[...]

7.3 Changes in [optional.syn]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— `bad_optional_access`
— `optional`

Please append a `//freestanding, partial` to the following entities:

— `optional`

7.4 Changes in [optional.optional.general]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— every reference qualified overload of `value`

Please append a `//freestanding, delete` to the following entities:

— every reference qualified overload of `value`

7.5 Changes in [variant.syn]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— every overload of `get`

Please append a `//freestanding, delete` to the following entities:

— every overload of `get`

7.6 Changes in [string.view.synop]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— `basic_string_view`
— `operator<<`

Please append a `//freestanding, partial` to the following entities:

— `basic_string_view`

7.7 Changes in [string.view.template.general]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— `at`
— `copy`
— `substr`
— The following overloads of `compare`:
  — `compare(size_type pos1, size_type n1, basic_string_view s)`
  — `compare(size_type pos1, size_type n1, basic_string_view s, size_type pos2, size_type n2)`
  — `compare(size_type pos1, size_type n1, const charT* s)`
  — `compare(size_type pos1, size_type n1, const charT* s, size_type n2)`

Please append a `//freestanding, delete` to the following entities:

— `at`
— `copy`
— `substr`
— The following overloads of `compare`:
  — `compare(size_type pos1, size_type n1, basic_string_view s)`
  — `compare(size_type pos1, size_type n1, basic_string_view s, size_type pos2, size_type n2)`
  — `compare(size_type pos1, size_type n1, const charT* s)`
  — `compare(size_type pos1, size_type n1, const charT* s, size_type n2)`

7.8 Changes in [array.syn]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— `array`

Please append a `//freestanding, partial` to the following entities:

— `array`

7.9 Changes in [array]

Instructions to the editor:

Please append a `//freestanding` to every entity except:

— `at`

Please append a `//freestanding, delete` to the following entities:

— `at`

7.10 Feature test macros

This part of the paper follows the guide lines as specified in [P2198R2]. Add the following macros to [version.syn]:

```
#define __cpp_lib_freestanding_array         20XXXXL //also in <array>
#define __cpp_lib_freestanding_optional      20XXXXL //also in <optional>
#define __cpp_lib_freestanding_string_view   20XXXXL //also in <string_view>
#define __cpp_lib_freestanding_variant       20XXXXL //also in <variant>
```

8 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/

[N4878] Thomas Köppe. 2020-12-15. Working Draft, Standard for Programming Language C++.
https://wg21.link/n4878

[P1642R5] Ben Craig. 2020-12-10. Freestanding Library: Easy [utilities], [ranges], and [iterators].
https://wg21.link/p1642r5

[P2198R2] Ben Craig. 2021-07-10. Freestanding Feature-Test Macros and Implementation-Defined Extensions.
https://wg21.link/p2198r2

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

[P2338R0] Ben Craig. 2021-03-13. Freestanding Library: Character primitives and the C library.
https://wg21.link/p2338r0