Remove Deprecated Locale Category Facets For Unicode from C++26

Document #:	P2873R1
Date:	2024-04-08
Project:	Programming Language C++
Audience:	LEWG

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net> Tom Honermann <tom@honermann.net>

Contents

1 Abstract

Several locale facets were added to C++11 that were intended to support Unicode transcoding. The facets that convert to and from UTF-8 were deprecated in C++20 when support for `char8_t` was added with the adoption of [P0482R6] due to their use of `char` for UTF-8 encoded data. This paper proposes removing those facets from the C++ Standard Library.

2 Revision History

R1: April 2024 (post-Tokyo mailing)

- Added Tom Honermann as coauthor
- Advanced to the next group for review: SG16 → LEWG
- Removed redundant subsection numbering from this section
- Revised the abstract and the "History" section
  - Noted that deprecation was motivated by the introduction of `char8_t`
  - Removed claims that deprecation was recommended by SG16; SG16 was formed after [P0482R6] was approved by EWG and LEWG
- Modified the "History" section to include additional details regarding the motivation for deprecation and the (incorrect) addition of `char8_t`-based replacements
- Added the "Deployment Experience" section with example code that would be expected to trigger a deprecation warning and updated the current implementation status accordingly
- Tested when (or if) popular Standard Library implementations warn of deprecation
- Recorded initial review feedback, recommending removal, from SG16
  - Added links to SG16 meeting summaries
- Revised wording
  - Validated wording against the latest working draft, [N4971]
  - Added Annex C wording
  - Updated stable label cross-reference to C++23
- Applied editorial feedback

R0: May 2023 (pre-Varna mailing)

- Initial draft of this paper

3 Introduction

At the start of the C++23 cycle, [P2139R2] tried to review each deprecated feature of C++ to see which would benefit us if actively removed and which might now be better undeprecated. Consolidating all this analysis into one place was intended to ease the (L)EWG review process but in return gave the author so much feedback that the next revision of the paper was not completed.

For the C++26 cycle, a much shorter paper, [P2863], will track the overall analysis, but for features that the author wants to actively progress, a distinct paper will decouple progress from the larger paper so that the delays on a single feature do not hold up progress on all.

This paper takes up the deprecated locale category facets for Unicode, D.24 [depr.locale.category].

4 History

The deprecated locale facets that convert to and from UTF-8 were added as part of the initial basic support for Unicode types in C++11 by paper [N2238]. However, they were introduced with inconsistent behavior relative to the corresponding `wchar_t` facet with respect to the encoding used for the `char`-typed side of the conversion. The `wchar_t` specialization converts to and from the locale dependent multibyte encoding while the (now deprecated) `char16_t` and `char32_t` facets convert to and from UTF-8. This behavior was an odd choice, not just for consistency but because the specified behavior is not affected by the choice of locale. Further, their addition was arguably unnecessary since the C++ Standard does not require support for streams of type `char16_t` or `char32_t`. (A previous revision of [N2238], [N2035] had proposed adding support for such streams, but that proposal was abandoned after the first revision based on LEWG feedback.)

At any rate, the result was two interfaces that associated different character encodings with type `char`. Prevention of this kind of encoding confusion was part of the motivation for the addition of `char8_t` in [P0482R6]. That paper deprecated these facets in favor of new ones that use `char8_t` for UTF-8.

Unfortunately, the [P0482R6] author failed to appreciate how the `std::codecvt` facets are actually intended to be used. Within the C++ Standard, they are used only by `std::basic_filebuf` (31.10.3 [filebuf]) and the `std::filesystem::path` constructor that accepts an argument of type `std::locale` (31.12.6.5.1 [fs.path.construct]) and are used only to convert streams or sequences of `wchar_t` to the sequence of `char` elements that are the contents or names of files. The new `char8_t`-based facets don't actually function as replacements because the existing use was fundamentally limited to use of (`char`-based) file streams and names. Since the Standard doesn't specify file streams of type `char8_t`, the newly added `char8_t`-based facets serve no actual purpose. (`std::filesystem::path` uses an unspecified conversion method for `char8_t`, `char16_t`, and `char32_t`; 31.12.6.3.2 [fs.path.type.cvt].) [LWG3767] now tracks their deprecation and future removal.

The status quo is thus as follows.

- The original addition of the deprecated facets was poorly motivated.
- The specified behavior of the deprecated facets is inconsistent and contrary to the purpose of such facets to provide locale dependent behavior.
- The facets have been deprecated since C++20.
- If the Standard were to add support for streams of type `char16_t` or `char32_t`, the existing behavior doesn't match the desired behavior (to convert to a locale dependent encoding).

5 C++23 Feedback

5.1 Initial LEWGI review: Telecon 2020/07/13

Discussion was broadly in favor of removal from the C++23 specification and reliance on library vendors to maintain source compatibility as long as needed. However, LEWGI explicitly requested the author to confer with SG16 to determine if that study group is aware of any reason to further delay removing this deprecated facility.

5.2 SG16 review: Telecon 2020/07/22

See https://github.com/sg16-unicode/sg16-meetings/blob/master/README-2020.md#july-22nd-2020.

SG16 is concerned that `std::codecvt` in general has poor error-handling facilities, especially when dealing with encodings that may take multiple code units to express a code point and thus have more cause to report on malformed inputs. The specific facets in D.24 [depr.locale.category] are an obstacle to putting a minimally useful replacement into the Standard because they reserve the good names but have poor semantics. The usual safety net of the zombie names clause does not apply since we will want code to fail to compile for at least one release to thus introduce a replacement with the same names but more appropriate semantics. Concern was raised about removing a feature deprecated only as recently as the current C++20 Standard.

Polling showed no consensus to recommend the removal for C++23 but no objection to that removal either.

5.3 LEWGI consensus

SG16 has no objection; remove this feature from C++23.

6 Proposal

Remove deprecated locale category facets for Unicode from C++26.

6.1 Deployment experience

The following example suffices for a deprecation warning to be emitted from popular current implementations when compiling in C++20 mode. See https://godbolt.org/z/5Yodo9KeW.

```
#include <locale>
struct user_codecvt_c16 : public std::codecvt<char16_t, char, std::mbstate_t> {};
struct user_codecvt_c32 : public std::codecvt<char32_t, char, std::mbstate_t> {};
struct user_codecvt_byname_c16 : public std::codecvt_byname<char16_t, char, std::mbstate_t> {};
struct user_codecvt_byname_c32 : public std::codecvt_byname<char32_t, char, std::mbstate_t> {};
```

— libc++: First warns in Clang 12 (released on 2021-04-14)
— libstdc++: Does not warn in the latest release
— MSVC: First warns with `/W3` or higher in MSVC v19.22 (VS 2019 version 16.2.3 released on 2019-08-20); does not warn by default or with `/W2` or lower in the latest release

7 C++26 Review

7.1 SG16 initial review: Telecon 2023/05/24

See https://github.com/sg16-unicode/sg16-meetings/blob/master/README-2023.md#may-24th-2023.

Tom Honermann explained the history and motivation behind the deprecation. SG16 then endorsed removing these facets from C++26.

7.2 SG16 second review: Telecon 2023/10/25

See https://github.com/sg16-unicode/sg16-meetings/blob/master/README-2023.md#october-25th-2023.

The proposal was reviewed again in light of [LWG3767], which deprecates the equivalent facets that convert to `char8_t` and were thought to be the intended replacement for the deprecated facets. We considered *undeprecating* these original facets as their replacement had not worked out.

However, further discussion demonstrated that the whole notion of using these facets, which are supposed to be locale dependent, to convert from one unicode encoding to another, using strict typing that ignores locale, was flawed. It was also observed that we are missing the facet to convert from `char8_t` to `char` if we wanted to continue maintaining these facets, and we do not want to be delving deeper.

There was a clear preference to remove these facets that are occupying the space in the library where a well specified replacement should go; we do not feel comfortable proving a replacement in that space until at least one Standard cycle has passed without these facets, minimizing the risk that code would silently change meaning when users update to a future standard.

Note that all discussion is predicated on a future library proposal to support unicode types in iostreams, where they are explicitly *not* supported today. Absent such a proposal, these facets serve no purpose, and are actively harmful due to their poor semantics.

The recommendation remains to pursue this paper as proposed.

8 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4971], the latest draft at the time of writing.

8.1 No changes to zombie names

All the entities being struck are overloads of identifiers that retain their original meaning, so no new names need to be added to 16.4.5.3.2 [zombie.names].

8.2 Add Annex C library wording

Add a new paragraph to C.1.7 [diff.cpp23.depr].

C.1.X Annex D: Compatibility features [diff.cpp23.depr]

Change: Remove `std::codecvt` locale facets that convert between `char16_t` or `char32_t` and `char`.

Rationale: The facets have been deprecated since C++ 2020 and have semantics that do not match their use as a locale-specific character conversion facility since they are specified to always convert to and from UTF-8 rather than the locale-specific multibyte encoding. Additionally, the C++ standard does not require support for streams of type `char16_t` or `char32_t`, so these locale facets need not be required by the C++ standard library.

Effect on original feature: A valid C++ 2023 program that uses any of the following template specializations may become ill-formed:

- `codecvt<char16_t, char, mbstate_t>`,
- `codecvt<char32_t, char, mbstate_t>`,
- `codecvt_byname<char16_t, char, mbstate_t>`, or
- `codecvt_byname<char32_t, char, mbstate_t>`

8.3 Strike library wording from Annex D

D.24 [depr.locale.category] Deprecated locale category facets

1 The `ctype` locale category includes the following facets as if they were specified in Table 104 of 30.3.1.2.1 [locale.category].

```
codecvt<char16_t, char, mbstate_t>
codecvt<char32_t, char, mbstate_t>
```

2 The ctype locale category includes the following facets as if they were specified in Table 105 of 30.3.1.2.1 [locale.category].

```
codecvt_byname<char16_t, char, mbstate_t>
codecvt_byname<char32_t, char, mbstate_t>
```

3 The following class template specializations are required in addition to those specified in 30.4.2.5 [locale.codecvt]. The specialization `codecvt<char16_t, char, mbstate_t>` converts between the UTF-16 and UTF-8 encoding forms, and the specialization `codecvt<char32_t, char, mbstate_t>` converts between the UTF-32 and UTF-8 encoding forms.

8.4 Update cross-reference for stable labels for C++23

Cross-references from ISO C++ 2023

All clause and subclause labels from ISO C++ 2023 (ISO/IEC 14882:2023, *Programming Languages — C++*) are present in this document, with the exceptions described below.

container.gen.reqmts *see*
24.2.2 [container.requirements.general]

depr.arith.conv.enum *removed*
depr.codecvt.syn *removed*
depr.default.allocator *removed*
depr.locale.category *removed*
depr.locale.stdcvt *removed*
depr.locale.stdcvt.general *removed*
depr.locale.stdcvt.req *removed*
depr.res.on.required *removed*
depr.string.capacity *removed*

mismatch *see* 27.6.12 [alg.mismatch]

9 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown.

Thanks to Lori Hughes for reviewing this paper and providing editorial feedback.

10 References

[LWG3767] Victor Zverovich. codecvt<charN_t, char8_t, mbstate_t> incorrectly added to locale.
https://wg21.link/lwg3767

[N2035] Matthew Austern. 2006-05-23. Minimal Unicode support for the standard library.
https://wg21.link/n2035

[N2238] Matthew Austern. 2007-04-17. Minimal Unicode support for the standard library (revision 3).
https://wg21.link/n2238

[N4971] Thomas Köppe. 2023-12-18. Working Draft, Programming Languages — C++.
https://wg21.link/n4971

[P0482R6] Tom Honermann. 2018-11-09. char8_t: A type for UTF-8 characters and strings (Revision 6).
https://wg21.link/p0482r6

[P2139R2] Alisdair Meredith. 2020-07-15. Reviewing Deprecated Facilities of C++20 for C++23.
https://wg21.link/p2139r2

[P2863] Alisdair Meredith. Review Annex D for C++26.
https://wg21.link/p2863