Remove Deprecated Unicode Conversion Facets from C++26
Removing an underspecified feature from C++26

Contents

1 Abstract

The `<codecvt>` header was first provided by C++11 and then deprecated by C++17 due to its under-specification, notably a lack of error handling. This paper proposes removing that header from the C++ Standard Library.

2 Revision History

R3: December 2023 (post-Kona mailing)

— Confirmed wording against latest Working Draft, [N4964]
— Updated compatibility wording to mention importing header units too
— Struck editors note as there will not be a race condition with another paper
— Removed remaining reference to `<codecvt>` header in diff to C++03

R2: September 2023 (midterm mailing)

— Removed revision history's redundant subsection numbering
— Wording updates
    — Rebased wording against latest Standard Working Draft, N4958
    — Updated stable label cross-reference to C++23
    — Close all open LWG issues on the removed feature
— Recorded reflector review
    — No objections but minor concerns about lack of replacement facility
    — Moved to electronic polling to advance to LWG

R1: 2023 August (midterm mailing)

— Recorded review feedback from SG16, recommending removal
— Added the test program used to determine which libraries warn of deprecation
— Updated test results to show MSVC has a deprecation warning at level `/W4`
— Removed `<codecvt>` from the table of headers in 16.4.2.3 [headers]
— Revised rationale in Annex C
— Validated wording against latest Standard Working Draft, N4950
— Assorted editorial details (thanks, Lori!)

R0: 2023 May (pre-Varna mailing)

Initial draft of the paper.

3 Introduction

At the start of the C++23 cycle, [P2139R2] tried to review each deprecated feature of C++ to see which we would benefit from actively removing 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, [P2863R2], 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 `<codecvt>` header in D.26 [depr.locale.stdcvt].

4 History of This Feature

The `<codecvt>` header was originally proposed for C++11 by paper [N2007] and deprecated for C++17 by paper [P0618R0]. As noted at the time of deprecation, this feature was underspecified and would require more work than we wished to invest to bring it up to the expected standard.

One particularly notable quote from that initial deprecation review was provided by Beman Dawes, who sought expert opinions from the domain, outside of WG21:

> As the Unicode standard and security experts point out, ill-formed UTF can be and has been used as an attack vector. Yet the `<codecvt>` facets don't provide the safe forms of error handling as their defaults, and it is far too hard to use them in a safe way.

Reports from users in the field indicated repeated attempts to use this facility, as a part of the Standard Library bearing the most useful name suggested this should be our best practice, only to see those attempts repeatedly fail for subtle reasons in our underspecification, leading to many homegrown solutions instead after much wasted time.

Since then, SG16 has convened and is producing a steady stream of work to bring reliable well-specified Unicode support to C++. However, no replacement facility is yet available as of C++23.

[P2736R2] provided a hasty fix for C++23 by removing the last reference to an antiquated Unicode standard that had been retained purely so that this deprecated C++ facility could refer to the obsolete-and-removed Unicode encoding, UCS-2.

5 Deployment Experience

The three main Standard Libraries provide these classes since they first supported C++11.

The following program, based on an example in the Standard, was tested with Godbolt Compiler Explorer to determine when (or if) libraries started warning about the deprecation.

```cpp
#include <codecvt>
#include <iostream>
#include <locale>
#include <string>

int main() {
   std::wstring_convert<std::codecvt_utf8<wchar_t>> myconv;
   std::string mbstring = myconv.to_bytes(L"Hello\n");
   std::cout << mbstring;
}
```

— libc++: First warns in Clang 15 (2022-09-06)
— libstdc++: Does not warn in latest release (gcc 13.2)

— MSVC: Warns with `/W4` since MSVC 19.14, oldest available through Godbolt

6 Proposal to Remove from C++26

This paper proposes removing the whole of the `<codecvt>` header from C++26, even without a replacement in place. Library vendors' freedom to provide ongoing support will be preserved by adding all relevant names to 16.4.5.3.2 [zombie.names].

As noted above, the deprecated feature has been an active nuisance in the past, if not an outright security risk when dealing with malformed Unicode. We have no plans to rehabilitate these facets, with SG16 working on a better approach to Unicode text processing in general.

If we remove this feature outright, existing user code is unlikely to break as vendors exercise their freedoms under the zombie names clause. However, new code is even less likely to use these poorly specified tools, and detractors will not be able to point to the C++ Standard.

When C++26 ships, this feature will have been deprecated for almost a decade and for three years longer than it was a part of the main library specification.

Note that the consensus of the review of this facility for C++23 was that the zombie names clause was sufficient support, and both SG16 and LEWG recommended removal in C++23. This removal did not occur as the author failed to update the whole Annex D review after [P2139R2].

6.1 SG16 review

SG16 reviewed this proposal in their regular telecon on 2023-05-24. Their recommendation, by unanimous consent, is to remove this feature.

Some concern initially arose for removing a feature before the replacement lands, but when the group realized that these names are inaccurate, as `_utf16` really means UCS-2, the desire to remove the header at the earliest opportunity became clear.

6.2 LEWG reflector review: 2023/08/14 − 2023/09/15

Several folks who did not object were still concerned that we would be removing a facility before a replacement lands; others acknowledged they were aware of this when polling to move forward and thought this was still the right choice. If we did not know that library vendors would be relying on the Zombie Names clause, we might have observed more objections.

Final tally: 13 forward to POLL, 1 Neutral-but-concerned.

Conclusion: Forward to the next electronic poll to send to LWG.

6.3 LWG Review: 2023 Kona meeting

Requested several improvements to the Compatibility Annex C wording, and then submitted for C++26.

7 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4964], the latest draft at the time of writing.

7.1 Impact of removal

This feature was originally added at the Kona 2007 meeting, updating the example in the then recently added [**lib.conversions**] (now D.27.2 [depr.conversions.string]) to use these Standard conversion facets rather than unspecified user-provided classes to illustrate use. Therefore, as we remove this library unilaterally, we must also amend that example to clearly rely on user-provided facets again.

The author has audited the current Standard for any remaining usage of the UCS-2 encoding removed below and has found none.

7.2 Strike `<codecvt>` from the list of standard headers

16.4.2.3 [headers] Headers

² The C++ standard library provides the *C++ library headers*, shown in Table 24.

Table 24: C++ library headers [tab:headers.cpp]

`<algorithm>`	`<forward_list>`	`<new>`	`<stdexcept>`
`<any>`	`<fstream>`	`<numbers>`	`<stdfloat>`
`<array>`	`<functional>`	`<numeric>`	`<stop_token>`
`<atomic>`	`<future>`	`<optional>`	`<streambuf>`
`<barrier>`	`<generator>`	`<ostream>`	`<string>`
`<bit>`	`<hazard_pointer>`	`<print>`	`<string_view>`
`<bitset>`	`<initializer_list>`	`<queue>`	`<strstream>`
`<charconv>`	`<iomanip>`	`<random>`	`<syncstream>`
`<chrono>`	`<ios>`	`<ranges>`	`<system_error>`
~~`<codecvt>`~~	`<iosfwd>`	`<ratio>`	`<text_encoding>`
`<compare>`	`<iostream>`	`<rcu>`	`<thread>`
`<complex>`	`<istream>`	`<regex>`	`<tuple>`
`<concepts>`	`<iterator>`	`<scoped_allocator>`	`<type_traits>`
`<condition_variable>`	`<latch>`	`<semaphore>`	`<typeindex>`
`<coroutine>`	`<limits>`	`<set>`	`<typeinfo>`
`<deque>`	`<list>`	`<shared_mutex>`	`<unordered_map>`
`<exception>`	`<locale>`	`<source_location>`	`<unordered_set>`
`<execution>`	`<map>`	`<span>`	`<utility>`
`<expected>`	`<mdspan>`	`<spanstream>`	`<valarray>`
`<filesystem>`	`<memory>`	`<sstream>`	`<variant>`
`<flat_map>`	`<memory_resource>`	`<stack>`	`<vector>`
`<flat_set>`	`<mutex>`	`<stacktrace>`	`<version>`
`<format>`			

7.3 Add new identifiers to 16.4.5.3.2 [zombie.names].

16.4.5.3.2 [zombie.names] Zombie names

¹ In namespace `std`, the following names are reserved for previous standardization:

— `auto_ptr`,
— `auto_ptr_ref`,
— `binary_function`,

- — binary_negate,
- — bind1st,
- — bind2nd,
- — binder1st,
- — binder2nd,
- — codecvt_mode,
- — codecvt_utf16,
- — codecvt_utf8,
- — codecvt_utf8_utf16,
- — const_mem_fun1_ref_t,
- — const_mem_fun1_t,
- — const_mem_fun_ref_t,
- — const_mem_fun_t,
- — consume_header,
- — declare_no_pointers,
- — declare_reachable,
- — generate_header,
- — get_pointer_safety,
- — get_temporary_buffer,
- — get_unexpected,
- — gets,
- — is_literal_type,
- — is_literal_type_v,
- — little_endian,
- — mem_fun1_ref_t,
- — mem_fun1_t,
- — mem_fun_ref_t,
- — mem_fun_ref,
- — mem_fun_t,
- — mem_fun,
- — not1,
- — not2,
- — pointer_safety,
- — pointer_to_binary_function,
- — pointer_to_unary_function,
- — ptr_fun,
- — random_shuffle,
- — raw_storage_iterator,
- — result_of,
- — result_of_t,
- — return_temporary_buffer,
- — set_unexpected,
- — unary_function,
- — unary_negate,
- — uncaught_exception,
- — undeclare_no_pointers,
- — undeclare_reachable, and
- — unexpected_handler.

2 The following names are reserved as members for previous standardization and may not be used as a name for object-like macros in portable code:

- — argument_type,
- — first_argument_type,
- — io_state,

- — `open_mode`,
- — `preferred`,
- — `second_argument_type`,
- — `seek_dir`, and.
- — `strict`.

[3] The name `stossc` is reserved as a member function for previous standardization and may not be used as a name for function-like macros in portable code.

[4] The header names `<ccomplex>`, `<ciso646>`, <u>`<codecvt>,`</u> `<cstdalign>`, `<cstdbool>`, and `<ctgmath>` are reserved for previous standardization.

7.4 Update Annex C

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

Change: Remove header `<codecvt>` and all its contents.

Rationale: The header has been deprecated for the previous three editions of this standard and no longer implements the current Unicode standard, supporting only the obsolete UCS-2 encoding. Ongoing support is at implementer's discretion, exercising freedoms granted by 16.4.5.3.2 [zombie.names].

Effect on original feature: A valid C++ 2023 program `#include`-ing the header or importing the header unit may fail to compile. Code that uses any of the following names by importing the standard library modules may fail to compile:

- — `codecvt_mode`
- — `codecvt_utf16`
- — `codecvt_utf8`
- — `codecvt_utf8_utf16`
- — `consume_header`
- — `generate_header`
- — `little_endian`

C.6.7 [diff.cpp03.library] Clause 16: library introduction

[2] **Affected subclause:** 16.4.2.3 [headers]

Change: New headers.

Rationale: New functionality.

Effect on original feature: The following C++ headers are new: `<array>` (24.3.2), `<atomic>` (33.5.2), `<chrono>` (29.2), ~~`<codecvt>` (D.26.2),~~ `<condition_variable>` (33.7.2), `<forward_list>` (24.3.4), `<future>` (33.10.2), `<initializer_list>` (17.10.2), `<mutex>` (33.6.2), `<random>` (28.5.2), `<ratio>` (21.4.2), `<regex>` (32.3), `<scoped_allocator>` (20.5.1), `<system_error>` (19.5.2), `<thread>` (33.4.2), `<tuple>` (22.4.2), `<typeindex>` (22.11.1), `<type_traits>` (21.3.3), `<unordered_map>` (24.5.2), and `<unordered_set>` (24.5.3). In addition the following C compatibility headers are new: `<cfenv>` (28.3.1),(31.13.2),(17.4.1), and' (23.5.5). Valid C++ 2003 code that #includes headers with these names may be invalid in this revision of C++.

7.5 Strike all of D.26 [depr.locale.stdcvt] Deprecated standard code conversion facets

D.26 [depr.locale.stdcvt] Deprecated standard code conversion facets

D.26.1 [depr.locale.stdcvt.general] General

[1] The header `<codecvt>` provides code conversion facets for various character encodings.

D.26.2 [depr.codecvt.syn] Header <codecvt> synopsis

```cpp
namespace std {
  enum codecvt_mode {
      consume_header = 4,
      generate_header = 2,
      little_endian = 1
  };

  template<class Elem, unsigned long Maxcode = 0x10ffff, codecvt_mode Mode = (codecvt_mode)0>
    class codecvt_utf8 : public codecvt<Elem, char, mbstate_t> {
    public:
      explicit codecvt_utf8(size_t refs = 0);
      ~codecvt_utf8();
    };

  template<class Elem, unsigned long Maxcode = 0x10ffff, codecvt_mode Mode = (codecvt_mode)0>
    class codecvt_utf16 : public codecvt<Elem, char, mbstate_t> {
    public:
      explicit codecvt_utf16(size_t refs = 0);
      ~codecvt_utf16();
    };

  template<class Elem, unsigned long Maxcode = 0x10ffff, codecvt_mode Mode = (codecvt_mode)0>
    class codecvt_utf8_utf16 : public codecvt<Elem, char, mbstate_t> {
    public:
      explicit codecvt_utf8_utf16(size_t refs = 0);
      ~codecvt_utf8_utf16();
    };
}
```

D.26.3 [depr.locale.stdcvt.req] Requirements

1   For each of the three code conversion facets `codecvt_utf8`, `codecvt_utf16`, and `codecvt_utf8_utf16`:

   — `Elem` is the wide-character type, such as `wchar_t`, `char16_t`, or `char32_t`.
   — `Maxcode` is the largest wide-character code that the facet will read or write without reporting a conversion error.
   — If `(Mode & consume_header)`, the facet shall consume an initial header sequence, if present, when reading a multibyte sequence to determine the endianness of the subsequent multibyte sequence to be read.
   — If `(Mode & generate_header)`, the facet shall generate an initial header sequence when writing a multibyte sequence to advertise the endianness of the subsequent multibyte sequence to be written.
   — If `(Mode & little_endian)`, the facet shall generate a multibyte sequence in little-endian order, as opposed to the default big-endian order.
   — UCS-2 is the same encoding as UTF-16, except that it encodes scalar values in the range u+0000–u+ffff (Basic Multilingual Plane) only.

2   For the facet `codecvt_utf8`:

   — The facet shall convert between UTF-8 multibyte sequences and UCS-2 or UTF-32 (depending on the size of `Elem`).
   — Endianness shall not affect how multibyte sequences are read or written.
   — The multibyte sequences may be written as either a text or a binary file.

3   For the facet `codecvt_utf16`:

   — The facet shall convert between UTF-16 multibyte sequences and UCS-2 or UTF-32 (depending on the

size of `Elem`).
— Multibyte sequences shall be read or written according to the `Mode` flag, as set out above.
— The multibyte sequences may be written only as a binary file. Attempting to write to a text file produces undefined behavior.

4   For the facet `codecvt_utf8_utf16`:

— The facet shall convert between UTF-8 multibyte sequences and UTF-16 (one or two 16-bit codes) within the program.
— Endianness shall not affect how multibyte sequences are read or written.
— The multibyte sequences may be written as either a text or a binary file.

7.6   Update example in D.27.2 [depr.conversions.string] Class template `wstring_convert`

D.27 [depr.conversions] Deprecated convenience conversion interfaces

D.27.2 [depr.conversions.string] Class template `wstring_convert`

1   Class template `wstring_convert` performs conversions between a wide string and a byte string. It lets you specify a code conversion facet (like class template `codecvt` (30.4.2.5.1 [locale.codecvt.general])) to perform the conversions, without affecting any streams or locales.

[*Example 1:* If you want to use ~~the~~a code conversion facet, `codecvt_for_utf8`, to output to `cout` a UTF-8 multibyte sequence corresponding to a wide string but you don't want to alter the locale for `cout`, you can write something like:

```
wstring_convert<std::codecvt_for_utf8<wchar_t>> myconv;
std::string mbstring = myconv.to_bytes(L"Hello\n");
std::cout << mbstring;
```

— *end example*]

```
namespace std {
  template<class Codecvt, class Elem = wchar_t,
           class WideAlloc = allocator<Elem>,
           class ByteAlloc = allocator<char>>
    class wstring_convert {
    public:
      // details elided
    };
}
```

2   The class template describes an object that …

7.7   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*
    container.requirements.general

depr.codecvt.syn *removed*
depr.locale.stdcvt *removed*
depr.locale.stdcvt.general *removed*
depr.locale.stdcvt.req *removed*
depr.res.on.required *removed*

7.8   Resolve open library issues

The following library issues should be resolved as NAD as they no longer apply to the C++ Standard due to the removal of the feature.

— [LWG2507] `codecvt_mode` should be a bitmask type

8 Acknowledgements

9 References

[LWG2507] Jonathan Wakely. codecvt_mode should be a bitmask type.
https://wg21.link/lwg2507

[N2007] P.J. Plauger. 2006-04-15. Proposed Library Additions for Code Conversion.
https://wg21.link/n2007

[N4964] Thomas Köppe. 2023-10-15. Working Draft, Programming Languages — C++.
https://wg21.link/n4964

[P0618R0] Alisdair Meredith. 2017-03-02. Deprecating <codecvt>.
https://wg21.link/p0618r0

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

[P2736R2] Corentin Jabot. 2023-02-09. Referencing the Unicode Standard.
https://wg21.link/p2736r2

[P2863R2] Alisdair Meredith. 2023-10-15. Review Annex D for C++26.
https://wg21.link/p2863r2