Remove Deprecated u8path overloads From C++26

Document #:	P3364R0
Date:	2024-07-22
Project:	Programming Language C++
Audience:	Library Evolution

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net>

Contents

1 Abstract **2**

2 Revision History **2**

3 Introduction **2**

4 Analysis **2**

5 C++23 Review History **3**
- 5.1 Initial LEWGI review: Telecon 2020/07/13 . . . 3
- 5.2 SG16 review: Telecon 2020/07/22 . . . 3
- 5.3 LEWG consensus (following SG16 review) . . . 3

6 Design Principles **4**

7 Proposed Solution **4**

8 C++26 Review History **4**
- 8.1 LEWG review: Telecon 2023/01/10 ([LWG3840]) . . . 4
- 8.2 SG16 review: Telecon 2023/05/24 ([P2863]) . . . 4
- 8.3 LEWG review: Kona, 2023/11/07 ([P2863]) . . . 4

9 Wording **5**
- 9.1 Add new identifiers to 16.4.5.3.2 [zombie.names] . . . 5
- 9.2 Update Annex C . . . 5
- 9.3 Strike from Annex D . . . 5
- 9.4 Update cross-reference for stable labels for C++23 . . . 6

10 Acknowledgements **8**

11 References **8**

1 Abstract

This paper proposes the removal of the deprecated `u8path` function from the next C++ Standard.

2 Revision History

R0 August 2024 (midterm mailing)

Initial draft of this paper, based on content in [P2863]

3 Introduction

The topic of this paper has been extracted from the general deprecation review paper, [P2863], into its own paper so as to better track its progress, since this topic has had a couple of reviews but is not reaching a real conclusion while embedded in the broader paper.

The `u8path` factory function creates path names from UTF-8 sequences of `char` and was part of the original filesystem library component adopted for C++17 via [P0218R1]. The function was deprecated in C++20 with the addition of `char8_t` to the core language and the ability to invoke a specific `path` constructor for UTF-8 encoded (and typed) strings; see [P0482R6] for details.

This paper proposes that this is the right time to remove the deprecated `u8path` function from the C++ Standard.

4 Analysis

The deprecated `u8path` function is now the only string-based factory function within the filesystem Standard Library component since the preferred interface has evolved to directly constructing a `path` with a string of the corresponding character type and its implied encoding. This legacy API continues to function but is more cumbersome than necessary.

No compelling case has been made that the API is a risk through misuse, although *the behavior is undefined* if fed malformed UTF-8. Even if the function appears to do no active harm, maintaining text in the Standard has a cost; for example, the original deprecated specification did not follow Library wording best practices and thus took up LWG review time to better specify the (also deprecated) *Requires:* clauses (see [P2874R2]).

The application of zombie names means that even if we remove this clause from Annex D in C++26, Standard Library vendors are likely to continue shipping this functionality to meet customer demand for some time to come.

5 C++23 Review History

This component was reviewed by telecon, achieving LEWG consensus for removal in C++23. However, the author ran out of time to complete the large paper handling *all* Annex D removals, and new information has since come to light with issue [LWG3840] requesting undeprecation of this function.

5.1 Initial LEWGI review: Telecon 2020/07/13

Discussion was broadly in favor of removing the `u8path` function from the C++23 specification and relying on library vendors to maintain source compatibility as long as needed. However, LEWGI explicitly requested, before proceeding with the recommendation, to confer with SG16 (Unicode) regarding any potential reason to hold back on removal.

5.2 SG16 review: Telecon 2020/07/22

SG16 was not persuaded that the removal of the `u8path` function is actually a text issue since filenames, in practice, accept a broader definition of text, and treating them as straight text is often a source of subtle and surprising problems. SG16 expressed mild concern at the idea of removing a function deprecated as recently as C++20 but also noted that the deprecating paper was adopted into the working draft before at least one major library distribution had provided its initial implementation. SG16 had no objection to the removal and offered mild encouragement to proceed.

5.3 LEWG consensus (following SG16 review)

Since SG16 raised no concerns, LEWG recommended removal of this feature from C++23.

6 Design Principles

Remove deprecated features from the Standard specification at the earliest *practical* opportunity to minimize the accumulation of technical debt.

7 Proposed Solution

Remove the deprecated Standard Library API from C++26 while granting vendors permission to continue supplying it as a conforming extension for as long as they desire through the use of zombie names.

8 C++26 Review History

8.1 LEWG review: Telecon 2023/01/10 ([LWG3840])

Discussion revealed little appetite for undeprecating `filesystem::u8path`. It is the only Standard Library interface that requires data provided in `char`-based storage to be encoded in a different character encoding than the execution character set or the encoding used for character and string literals.

For those who intentionally store UTF-8 data in `char`-based storage and would prefer not to use a deprecated interface, an equivalent to `filesystem::u8path` can be implemented in a few lines of code:

```
inline auto myu8path(const char* s) {
  std::u8string u8s(s, s+std::strlen(s));
  return std::filesystem::path(u8s);
}
```

SG16, the Unicode and Text Study Group, is evaluating approaches to enable restricted aliasing support for `char8_t` such that data in char-based storage could be passed to a `char8_t`-based interface without having to perform a copy, such as the one shown in the example code above; [P2626] is one such proposal.

8.2 SG16 review: Telecon 2023/05/24 ([P2863])

SG16 observed that [P2626] aims to solve the root issue behind this problem and that we should not remove a feature until it has coexisted with the facility to which to migrate for at least one Standard release. SG16 showed no enthusiasm for undeprecation and a general leaning toward removal in due course, maybe C++29. SG16 strongly recommended abiding by the status quo on this function for C++26.

The observation was made that the original motivation for deprecation was to dissuade continuing to provide Standard Library functions that require UTF-8 data in `char`-based storage. This function was one of just two Standard Library features that did so, and the other has since been recommended for removal by [P2873R2].

8.3 LEWG review: Kona, 2023/11/07 ([P2863])

Some LEWG members mentioned using the `u8path` function and the awkwardness of writing code that is portable across multiple Standards and that does not rely on deprecated features; that problem would go away if we undeprecated this API, which behaves exactly as they expect.

Conversely, others in LEWG would like to remove a potential source of confusion now that we have a type-safe interface that properly reflects the text encoding.

An observation was made that gaining consensus in either direction — undeprecation or removal — might be difficult and that this feature might remain in limbo (Annex D) for many Standards.

Since the goal of [P2863] is to move deprecated features out of limbo, LEWG did reach consensus to send this feature back to SG16 asking for their preferred direction for the long-term future of this function.

9 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4986], the latest draft at the time of writing.

9.1 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 names shown in Table 38 are reserved for previous standardization:

Table 1: Table 38 — Zombie names in namespace std [tab:zombie.names.std]

auto_ptr	~~generate_header~~	~~pointer_to_binary_function~~
auto_ptr_ref	get_pointer_safety	pointer_to_unary_function
binary_function	get_temporary_buffer	ptr_fun
binary_negate	get_unexpected	random_shuffle
bind1st	gets	raw_storage_iterator
bind2nd	is_literal_type	result_of
binder1st	is_literal_type_v	result_of_t
binder2nd	istrstream	return_temporary_buffer
codecvt_mode	little_endian	set_unexpected
codecvt_utf16	mem_fun1_ref_t	strstream
codecvt_utf8	mem_fun1_t	strstreambuf
codecvt_utf8_utf16	mem_fun_ref_t	u8path
const_mem_fun1_ref_t	mem_fun_ref	unary_function
const_mem_fun1_t	mem_fun_t	unary_negate
const_mem_fun_ref_t	mem_fun	uncaught_exception
const_mem_fun_t	not1	undeclare_no_pointers
consume_header	not2	undeclare_reachable
declare_no_pointers	ostrstream	unexpected_handler
declare_reachable	pointer_safety	wbuffer_convert
generate_header	pointer_to_binary_function	wstring_convert

9.2 Update Annex C

STILL TO PROVIDE WORDS FOR ANNEX C

9.3 Strike from Annex D

[depr.fs.path.factory] Deprecated filesystem path factory functions

1 The header `<filesystem>` (31.12.4 [fs.filesystem.syn]) has the following additions:

```
template<class Source>
  path u8path(const Source& source);
template<class InputIterator>
  path u8path(InputIterator first, InputIterator last);
```

2 *Mandates:* The value type of `Source` and `InputIterator` is `char` or `char8_t`.

3 *Preconditions:* The `source` and [`first`, `last`) sequences are UTF-8 encoded. `Source` meets the requirements specified in 31.12.6.4 [fs.path.req].

4 *Returns:*

— If `path::value_type` is `char` and the current native narrow encoding (31.12.6.3.2 [fs.path.type.cvt]) is UTF-8, return `path(source)` or `path(first, last)`; otherwise,
— if `path::value_type` is `wchar_t` and the native wide encoding is UTF-16, or if `path::value_type` is `char16_t` or `char32_t`, convert `source` or [`first`, `last`) to a temporary, `tmp`, of type `path::string_type` and return `path(tmp)`; otherwise,
— convert `source` or [`first`, `last`) to a temporary, `tmp`, of type `u32string` and return `path(tmp)`.

5 *Remarks:* Argument format conversion (31.12.6.3.1 [fs.path.fmt.cvt]) applies to the arguments for these functions. How Unicode encoding conversions are performed is unspecified.

6 [*Example 1:* A string is to be read from a database that is encoded in UTF-8, and used to create a directory using the native encoding for filenames:

```
namespace fs = std::filesystem;
std::string utf8_string = read_utf8_data();
fs::create_directory(fs::u8path(utf8_string));
```

For POSIX-based operating systems with the native narrow encoding set to UTF-8, no encoding or type conversion occurs.

For POSIX-based operating systems with the native narrow encoding not set to UTF-8, a conversion to UTF-32 occurs, followed by a conversion to the current native narrow encoding. Some Unicode characters may have no native character set representation.

For Windows-based operating systems a conversion from UTF-8 to UTF-16 occurs. —*end example*]

[*Note 1:* The example above is representative of a historical use of `filesystem::u8path`. To indicate a UTF-8 encoding, passing a `std::u8string` to `path`'s constructor is preferred as it is consistent with `path`'s handling of other encodings. —*end note*]

—*end note*]

9.4 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*
container.requirements.general

depr.arith.conv.enum *removed*
depr.codecvt.syn *removed*
depr.conversions *removed*
depr.conversions.buffer *removed*
depr.conversions.general *removed*
depr.conversions.string *removed*
depr.default.allocator *removed*
depr.fs.path.factory *removed*
depr.istrstream *removed*
depr.istrstream.cons *removed*
depr.istrstream.general *removed*
depr.istrstream.members *removed*
depr.locale.stdcvt *removed*
depr.locale.stdcvt.general *removed*
depr.locale.stdcvt.req *removed*
depr.mem.poly.allocator.mem *see*

mem.poly.allocator.mem
depr.ostrstream *removed*
depr.ostrstream.cons *removed*
depr.ostrstream.general *removed*
depr.ostrstream.members *removed*
depr.res.on.required *removed*
depr.string.capacity *removed*
depr.str.strstreams *removed*
depr.strstream *removed*
depr.strstream.cons *removed*
depr.strstream.dest *removed*
depr.strstream.general *removed*
depr.strstream.oper *removed*
depr.strstream.syn *removed*
depr.strstreambuf *removed*
depr.strstreambuf.cons *removed*
depr.strstreambuf.general *removed*
depr.strstreambuf.members *removed*
depr.strstreambuf.virtuals *removed*
depr.util.smartptr.shared.atomic *removed*

mismatch *see* alg.mismatch

10 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.

11 References

[LWG3840] Daniel Krügler. filesystem::u8path should be undeprecated.
https://wg21.link/lwg3840

[N4986] Thomas Köppe. 2024-07-16. Working Draft, Programming Languages — C++.
https://wg21.link/n4986

[P0218R1] Beman Dawes. 2016-03-05. Adopt File System TS for C++17.
https://wg21.link/p0218r1

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

[P2626] Corentin Jabot. `charN_t` incremental adoption: Casting pointers of UTF character types.
https://wg21.link/p2626

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

[P2873R2] Alisdair Meredith, Tom Honermann. 2024-07-06. Remove Deprecated locale category facets for Unicode from C++26.
https://wg21.link/p2873r2

[P2874R2] Alisdair Meredith. 2023-06-12. Mandating Annex D.
https://wg21.link/p2874r2