`define_static_string` as a `STATICALLY_WIDEN` replacement

Document #:	P3687R1
Date:	2026-10-01
Programming Language C++	
Audience:	SG-16

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

We propose a reflection-inspired facility to replace the `STATICALLY-WIDEN` utility macro.

Revisions

R1

Note that an implementation is now available.

Motivation

The `STATICALLY-WIDEN` magic utility is defined ([time.general]) as

`Let STATICALLY-WIDEN<charT>("...") be "..." if charT is char and L"..." if charT is wchar_t.`

This can only be implemented using macro tricks. This is a possible implementation based on the one found in libc++:

```
template <typename _CharT>
constexpr const CharT* __statically_widen(const char* str, const wchar_t* wstr) {
    if constexpr (same_as<_CharT, char>)
      return __str;
    else
      return __wstr;
}
#define STATICALLY_WIDEN(CharT, str) statically_widen<CharT>(str, L##str)
```

The obvious issue is that it can only be used with string literals, as it depends on preprocessing.

We proposed to expose that magic facility to users, using reflection and compiler support, to support any compile-time value (not just string literals)

Use cases

Interfaces that are designed to work with multiple character types (char, char8_t, wchar_t, etc) need to provide strings for default values, format strings, localization strings, etc. However,

the type of literal is determined during lexing, and there exist no compile-time transcoding facilities today.

Design

We propose to add a `define_encoded_static_string` function, modeled on `define_static_string` taking a range of UTF-8 code units as a parameter, and producing a string literal in the associated encoding of an explicitly specified template parameter

```
const char* hello = define_encoded_static_string<char>(u8"Hello");
const wchar_t* wello = define_encoded_static_string<wchar_t>(u8string_view(u8"Hello"));
```

Input type

We picked char8_t as the input type because UTF-8 is the usual way to represent Unicode data. (and the feature would not be very useful if the uinput value could not represent all characters).

Supporting an arbitrary character type for the input did not seem that useful; however, it would not add much work to do so.

Alternatives considered

Why not making generalized transcoding facilities `constexpr`?

The easy answer is that such facilities do not exist yet. But even then, to expose something more generic capable of performing arbitrary transcoding, the compiler would have to expose a bigger set of builtins to expose iconv or ICU. This would require a much bigger implementation effort, and there would be questions about the portability of various encodings.

There is not that much motivation to support at compile-time encodings beyond the literal encodings used by the standard. Sure, I'm sure someone could come up with some use case (that would be better served by a build script), either way, it doesn't seem like it would be a good use of implementer time, so we should prefer a pragmatic, more focused solution for the 99% use case.

`__LPREFIX`

`__LPREFIX` is a MSVC-prefix macro to statistically widen macros like `__FUNNCTION__`. It suffers the same limitation as `STATICALLY_WIDEN` and has also proven to be extremely challenging to implement in Clang.

Implementation

This proposal has been implemented in the reflection fork of clang maintained by Bloomberg, and is available on Compiler Explorer. The goal of the implementation was to demonstrate

that it requires minimal efforts as this reuse pieces that already need to exist in a compiler.

Wording

[meta.define.static] Promoting to static storage strings

The functions in this subclause promote compile-time storage into static storage.

```
template<ranges::input_range R>
consteval const ranges::range_value_t<R>* define_static_string(R&& r);
```

Effects: Equivalent to:

```
return extract<const ranges::range_value_t<R>*>(meta::reflect_constant_string(r));
```

```
template<typename CharT, ranges::input_range R>
requires same_as<char8_t, ranges::range_value_t<R>>
consteval const CharT* define_encoded_static_string(R&& r);
```

Mandates: `CharT` is one of `char`, `wchar_t`, `char8_t`, `char16_t`, `char32_t`.

Let Seq be a sequence of `char8_t` code units whose elements are the corresponding elements of `r`, except that if `r` refers to a string literal object, then Seq does not include the trailing null terminator of `r`.

Seq is converted to a sequence Seq' of `CharT` in the associated character encoding of `CharT` [lex.ccon]. If a character lacks representation in the associated character encoding, or if r is not a valid UTF-8 code unit sequence, then the call to `define_encoded_static_string` is not a constant expression.

Effects: Returns a string literal object formed from each successive element in Seq' followed by `CharT()`.

Acknowledgments

I'd like to thanks Dan Katz for his work on the original implementation of P2996 in Clang, The folks at Bloomberg who maintain that fork and let merge the implementation of this proposal, and the Compiler Explorer team for graciously hosting it.

References