What's in a `cstring_view`?

Document #:	P4227R0
Date:	2026-05-11
Project:	Programming Language C++ Library Evolution Working Group

Reply-to:	Andreas Weis <cpp@andreas-weis.net>

Abstract

[P3655] proposes `std::cstring_view` as a string-view type for null-terminated strings. In its current revision, the proposed type supports embedded NUL bytes. This paper argues why that is not a good design choice for such a type.

1 Introduction

The motivation for `cstring_view` is interaction with "operating system calls, C interfaces, third-party library APIs, or even C++ standard library APIs which require null-terminated strings"[P3655]. The paper argues in Section 6.4 that allowing embedded NUL bytes is reasonable, because some C APIs do support embedded NULs in input strings and disallowing them would make `cstring_view` unusable for interacting with those APIs.

This paper attempts to make an argument why the proposed support for embedded NUL bytes severely reduces the usefulness of the type and may hinder adoption of `cstring_view` for its primary use case of interoperability with C APIs.

2 C Strings end at the first NUL byte

In C, a `'\0'` in a char array marks the end of the string. A C string is not just a string that has a `'\0'` at the end. It's a string that *ends at the first `'\0'` character*.

In particular a `cstring_view` with embedded NUL bytes will cause friction in all contexts that rely on either the size, or the position of the end of the string.

Consider the following examples:

```
FILE* open_txt_file(std::cstring_view filename) {
  if (filename.ends_with(".txt")) {
    return fopen(filename.c_str(), "rw");
  } else {
    auto const filename_with_ext = std::string{ filename } + ".txt";
    return fopen(filename_with_ext.c_str(), "rw");
  }
}
```

This function will not work correctly with inputs like `"hello\0.txt"`. The reason is that the check inside the `if` has a different notion of the end of the string than `fopen`.

It is likely that every use of `cstring_view`'s `ends_with` will be an instance of such a logic bug. If `cstring_view` allows embedded NUL bytes, it probably should not have an `ends_with` function.

```
bool check_and_set_password(std::cstring_view password) {
  if (password.size() < 15) {
    // password is too short
    return false;
  }
  set_password(password.c_str());
  return true;
}
```

This function is supposed to reject passwords that are too short, but will accept inputs like `"123\0I-tricked-you!Hahaha!"`.

One might argue that, in practice, crypto APIs like OpenSSL's `EVP_DigestUpdate()` commonly require passing the size explicitly. But such APIs also do not require a terminating `'\0'` at the end of the string, so `string_view` or even `span` are suitable for interacting with such APIs. If a user is dealing with an API that requires the use of `cstring_view`, they will be susceptible to this kind of bug.

```
char* create_string_copy(std::cstring_view source_string) {
  char* new_string = malloc(source_string.size());
  strcpy(new_string, source_string.c_str());
  return new_string;
}
```

This function will allocate too much memory for its intended purpose. To mitigate this, an implementation must not rely on `cstring_view`'s `size()` function. A function which, according to P3655, is one of the key motivations for introducing this type in the first place.

Conceptually, C APIs don't consider anything past the first NUL byte to be part of the string. A type whose primary purpose is interacting with such APIs should have the same notion of the contents of a string.

3 But `string` and `string_view` have all of these problems as well!

They do.

But unlike `cstring_view`, they were not designed for interop with C functions. Consistency with the existing string types is at odds with allowing smooth interop with C APIs.

4 But what if I have a C API that requires strings with embedded NULs?

P3655 acknowledges that C APIs that allow embedded NUL bytes are rare.

The author is aware of two types of C APIs that are able to process such strings correctly.

1. APIs accepting an explicit `size` argument. Such APIs do not rely on the terminating NUL byte to detect the end of the string. However, because of this, they also do not *require* the string to be null terminated. `string_view` and `span` already sufficient for interacting with such APIs.
2. APIs that require "double-null-terminated strings". Such strings are commonly used in the Win32 API[1][2], but are also rarely encountered in other places[3]. A double-null-terminated string is really a list of strings, where each individual string is terminated by a

single NUL byte and then finally a sequence of two NUL bytes terminating the entire list. However, `cstring_view` is not suitable for use with such APIs, because they require the input string to be terminated by *two* NUL bytes, while `cstring_view` only guarantees a single terminator at the end.

While there may be use cases for a type that represents a string that is both null-terminated *and* may contain embedded NUL bytes, those uses are much rarer and sufficiently different that, if supporting them through the standard library is desired, they should be handled by a different library type.

For example a `cezstring_view` (pronounced *see-zee*, rhymes with sleazy, because it kind of is) could be added alongside `cstring_view` for this purpose. Such a type would primarily address the needs of users that are concerned about the runtime overhead required to enforce the absence of embedded NUL bytes for `cstring_view`. Such a type should omit the problematic `ends_with` function. The distinction between `cstring_view` and `cezstring_view` makes it clear which properties regarding its size a user can rely on when passing a value on to a C API.

References

[P3655] *P3655R4 - `std::cstring_view`*
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2026/p3655r4.html

[1] *`REG_MULTI_SZ` Registry value type*
https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry-value-types

[2] *`lpstrFilter` field in `OPENFILENAMEA` struct*
https://learn.microsoft.com/en-us/windows/win32/api/commdlg/ns-commdlg-openfilenamea

[3] *proc_pid_environ(5) — Linux manual page*
https://man7.org/linux/man-pages/man5/proc_pid_environ.5.html