

# Range constructor for `std::string_view`

Document #: D1391R1  
Date: 2019-06-07  
Project: Programming Language C++  
Audience: LEWG  
Reply-to: Corentin Jabot <[corentin.jabot@gmail.com](mailto:corentin.jabot@gmail.com)>

## 1 Abstract

This paper proposes that `string_view` be constructible from any contiguous range of characters. The idea was extracted from P1206.

## 2 Tony tables

Before	After
<pre>void foo(string_view); vector&lt;char8_t&gt; vec = get_some_unicode(); foo(string_view{vec.data(), vec.size()});</pre>	<pre>void foo(string_view); vector&lt;char8_t&gt; vec = get_some_unicode(); foo(vec);</pre>

## 3 Motivation

While P1206 gives a general motivation for range constructors, it's especially important for `string_view` because there exist in a lot of codebases string types that would benefit from being convertible to `string_view`. For example, `llvm::StringRef`, `QByteArray`, `fbstring`, `boost::container::string` ...

Manipulating the content of a vector as a string is also useful.

Finally, this makes contiguous views operating on characters easier to use with `string_view`.

## 4 Design considerations

- instantiations of `basic_string` are specifically excluded because `std::basic_string` already provides a conversion operator and more importantly, strings with different `char_traits` should not be implicitly convertible
- Because `basic_string_view` doesn't mutate the underlying data, there is no reason to accept a range by something other than const lvalue reference.

- The construction is implicit because it is cheap and a contiguous range of character is the same platonic thing as a `string_view`.

## 5 Proposed wording

Change in `[string.view]` 20.4.2:

```
template<class charT, class traits = char_traits<charT>>
class basic_string_view {
public:
    [...]

    // construction and assignment
    constexpr basic_string_view() noexcept;
    constexpr basic_string_view(const basic_string_view&) noexcept = default;
    constexpr basic_string_view& operator=(const basic_string_view&) noexcept = default;
    constexpr basic_string_view(const charT* str);
    constexpr basic_string_view(const charT* str, size_type len);

    template <typename R>
    constexpr basic_string_view(const R& r);

    template <typename It, typename End>
    constexpr basic_string_view(It begin, End end);

    [...]
};
```

Change in `[string.view.cons]` 20.4.2.1:

Add after 7

```
template <typename R>
constexpr basic_string_view(const R& r);
```

*Constraints:*

- `R` models `ranges::ContiguousRange`.
- `ranges::SizedRange<const R>` is true.
- `Same<iter_value_t<iterator_t<const R>>, charT>` is true.
- `is_convertible_v<const R&, const charT*>` is false.
- `R` does not derive from a specialization of `std::basic_string`.
- `R` does not derive from a specialization of `std::basic_string_view`.

*Effects:* Constructs a `basic_string_view` over the `ContiguousRange` `r` by initializing `data_` with `ranges::data(r)` and `size_` with `ranges::size(r)`.

*Throws:* What and when `ranges::data(r)` and `ranges::size(r)` throw.

```
template <typename It, typename End>  
constexpr basic_string_view(It begin, End end);
```

*Constraints:*

- It models `ContiguousIterator`.
- End models `SizedSentinel<It>`.
- `Same<iter_value_t<It>, charT>` is true.
- It does not derive from an instantiation of `std::basic_string::iterator` or `std::basic_string::const_iterator`.
- It does not derive from an instantiation of `std::basic_string_view::iterator` or `std::basic_string_view::const_iterator`.
- It and End are not of the same type or End is not convertible to a pointer of `charT`.

*Expects:* `[begin, end)` shall be a valid range.

*Effects:* Constructs a `basic_string_view` over the range `[begin, end)` by initializing `data_` with `data(begin)` and `size_` with `distance(begin, end)`.