Interfacing stringstream with string_view

Abstract
This paper proposes amending the interface of basic_[i|o]stringstream and basic_stringbuf to support construction and reinitialization from basic_string_view. As a drive-by-fix it also enables construction from raw string when supplying an allocator.

Tony Table

<table>
<thead>
<tr>
<th>Before</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>const ios_base::openmode mode;</td>
<td>const ios_base::openmode mode;</td>
</tr>
<tr>
<td>const allocator&lt;char&gt; alloc;</td>
<td>const allocator&lt;char&gt; alloc;</td>
</tr>
<tr>
<td>const string str;</td>
<td>const string str;</td>
</tr>
<tr>
<td>//implicitly convertible to string_view</td>
<td>//implicitly convertible to string_view</td>
</tr>
<tr>
<td>const mystring mstr;</td>
<td>const mystring mstr;</td>
</tr>
<tr>
<td>stringstream s0(&quot;&quot;);</td>
<td>stringstream s0(&quot;&quot;);</td>
</tr>
<tr>
<td>stringstream s1(&quot;&quot;, alloc);</td>
<td>stringstream s1(&quot;&quot;, alloc);</td>
</tr>
<tr>
<td>stringstream s2(&quot;&quot;, mode, alloc);</td>
<td>stringstream s2(&quot;&quot;, mode, alloc);</td>
</tr>
<tr>
<td>stringstream s3(&quot;sv&quot;);</td>
<td>stringstream s3(&quot;sv&quot;);</td>
</tr>
<tr>
<td>stringstream s4(&quot;sv&quot;, alloc);</td>
<td>stringstream s4(&quot;sv&quot;, alloc);</td>
</tr>
<tr>
<td>stringstream s5(&quot;sv&quot;, mode, alloc);</td>
<td>stringstream s5(&quot;sv&quot;, mode, alloc);</td>
</tr>
<tr>
<td>stringstream s6(&quot;s&quot;);</td>
<td>stringstream s6(&quot;s&quot;);</td>
</tr>
<tr>
<td>stringstream s7(&quot;s&quot;, alloc);</td>
<td>stringstream s7(&quot;s&quot;, alloc);</td>
</tr>
<tr>
<td>stringstream s8(&quot;s&quot;, mode, alloc);</td>
<td>stringstream s8(&quot;s&quot;, mode, alloc);</td>
</tr>
<tr>
<td>stringstream s9(str);</td>
<td>stringstream s9(str);</td>
</tr>
<tr>
<td>stringstream s10(str, alloc);</td>
<td>stringstream s10(str, alloc);</td>
</tr>
<tr>
<td>stringstream s11(str, mode, alloc);</td>
<td>stringstream s11(str, mode, alloc);</td>
</tr>
<tr>
<td>stringstream s12(mstr);</td>
<td>stringstream s12(mstr);</td>
</tr>
<tr>
<td>stringstream s13(mstr, alloc);</td>
<td>stringstream s13(mstr, alloc);</td>
</tr>
<tr>
<td>stringstream s14(mstr, mode, alloc);</td>
<td>stringstream s14(mstr, mode, alloc);</td>
</tr>
</tbody>
</table>

Revisions
R0: Initial version

---

1 RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg, Austria, michael.hava@risc-software.at
Motivation

[string.view] specifies basic_string_view, a vocabulary type template that represents an immutable reference to some string-like object. Unless a string can be moved from source to target, it is generally advisable to pass “immutable stringy inputs” by basic_string_view. Doing so obviates the need for multiple overloads and enables support for user-defined types.

[string.streams] specifies the class templates basic_i|ostringstream and basic_stringbuf to represent streams operating on/buffers owning a string. These classes predate the introduction of basic_string_view and therefore only support basic_string in their interfaces. Partial support for raw strings is provided by implicitly constructing a basic_string and then moving it.

This leads to an embarrassing problem when following the aforementioned recommendation: Every basic_string_view and user-defined string type must be explicitly converted to a temporary basic_string that is then moved into the respective constructor/member function. This paper aims to solve these issues by introducing direct support for basic_string_view.

Design space

As all classes in [string.streams] adhere to the following fragment for the context of construction/reinitialization from a string, the potential design is presented in terms of CLASS:

```
template<typename CharT, typename Traits, typename Alloc>
struct CLASS {
  // constructors interfacing with stringy inputs
  explicit CLASS(const basic_string<CharT, Traits, Alloc>&, ios_base::openmode = /*def*/);

  template<typename SAlloc>
  CLASS(const basic_string<CharT, Traits, SAlloc>&, const Alloc&);

  template<typename SAlloc>
  CLASS(const basic_string<CharT, Traits, SAlloc>&, ios_base::openmode, const Alloc&);

  template<typename SAlloc>
  requires(!std::is_same_v<SAlloc, SAlloc>)
  explicit CLASS(basic_string<CharT, Traits, Alloc>&&, ios_base::openmode = /*def*/);

  //reinitialization of internal string
  void str(const basic_string<CharT, Traits, Alloc>&);

  template<typename SAlloc>
  requires(!std::is_same_v<SAlloc, SAlloc>)
  void str(const basic_string<CharT, Traits, SAlloc>&);

  void str(basic_string<CharT, Traits, Alloc>&&);
};
```

The constructor and member function overloads can roughly be classified as follows:

<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Copying the string.</td>
</tr>
<tr>
<td>2</td>
<td>Copying the string, input may have different allocator. Invalid for const CharT *.</td>
</tr>
<tr>
<td>3</td>
<td>Copying the string.</td>
</tr>
<tr>
<td>4</td>
<td>Equal to 1 but input has different allocator. Invalid for const CharT *.</td>
</tr>
<tr>
<td>5</td>
<td>Moving the string, used for const CharT *.</td>
</tr>
<tr>
<td>6</td>
<td>Copying the string.</td>
</tr>
<tr>
<td>7</td>
<td>Equal to 6 but input has different allocator. Invalid for const CharT *.</td>
</tr>
<tr>
<td>8</td>
<td>Moving the string, used for const CharT *.</td>
</tr>
</tbody>
</table>
There are several possible designs to add support for basic_string_view, some of which also fix the unsupported construction from const CharT * in [2][3].

**Option 1: Add additional overloads**

Introduce basic_string_view-overloads for [1][2][3][6]. The overloads to [1][6] need special treatment to resolve an ambiguity as both basic_string and basic_string_view are implicitly constructible from const CharT *, for which there are three possible approaches:

**Option 1a: Additional raw string overloads**

Introduce dedicated const CharT * overloads for [1][6] – overload resolution for existing usages of const CharT * changes from [5][8] to the new semantically equivalent overloads.

```cpp
//add to existing class definition:
explicit CLASS(const CharT *, ios_base::openmode = /*def*/);
explicit CLASS(basic_string_view<CharT, Traits>, ios_base::openmode = /*def*/);
CLASS(basic_string_view<CharT, Traits>, const Alloc&);
CLASS(basic_string_view<CharT, Traits>, ios_base::openmode, const Alloc&);
void str(const CharT *);
void str(basic_string_view<CharT, Traits>);
```

**Option 1b: Restricting new overloads**

Restrict the new overloads of [1][6]; handling of a single const CharT * is unchanged. Parameters implicitly convertible to basic_string_view are not supported by this approach!

```cpp
//add to existing class definition:
explicit CLASS(same_as<basic_string_view<CharT, Traits>>, auto, ios_base::openmode = /*def*/);
CLASS(basic_string_view<CharT, Traits>, const Alloc&);
CLASS(basic_string_view<CharT, Traits>, ios_base::openmode, const Alloc&);
void str(same_as<basic_string_view<CharT, Traits>> auto);
```

**Option 1c: Restricting existing overloads**

Restrict the existing [1][5][6][8]; const CharT * is handled by the overloads of [1][6]. Depending on the implementation strategy, this might result in an ABI break.

```cpp
//add to existing class definition:
explicit CLASS(basic_string_view<CharT, Traits>, ios_base::openmode = /*def*/);
CLASS(basic_string_view<CharT, Traits>, const Alloc&);
CLASS(basic_string_view<CharT, Traits>, ios_base::openmode, const Alloc&);
void str(basic_string_view<CharT, Traits>);

//add constraints to existing constructors/member functions:
explicit CLASS(const same_as<basic_string<CharT, Traits, Alloc>> auto&, ios_base::openmode = /*def*/);
explicit CLASS(same_as<basic_string<CharT, Traits, Alloc>> auto&&, ios_base::openmode = /*def*/);
void str(const same_as<basic_string<CharT, Traits, Alloc>> auto&);
void str(same_as<basic_string<CharT, Traits, Alloc>> auto&&);
```
Option 2: Replace existing constructors and member functions
Switch the first parameter type of to basic_string_view; remove as they are no longer needed; restrict to disambiguate handling of const * with . These extensive changes result in an ABI break.

```cpp
// replace existing constructors/member functions:
explicit CLASS(basic_string_view<CharT, Traits>, ios_base::openmode = /*def*/);
CLASS(basic_string_view<CharT, Traits>, const Alloc&);
CLASS(basic_string_view<CharT, Traits>, openmode, const Alloc&);
explicit CLASS(same_as<basic_string<CharT, Traits, Alloc>> auto&&, ios_base::openmode = /*def*/);
void str(basic_string_view<CharT, Traits>);
void str(same_as<basic_string<CharT, Traits, Alloc>> auto&&);
```

Recommendation
We propose Option 1a as it enables the functionality whilst avoiding ABI breaks. The selected design changes overload resolution for raw strings, but the resulting overload is semantically equivalent to the existing behavior.

Impact on the Standard
This proposal is a pure library addition. Existing standard library classes are modified in a non-ABI-breaking way.

Implementation Experience
All options have been implemented on [https://godbolt.org/z/49xdWEKG4](https://godbolt.org/z/49xdWEKG4) to evaluate the resulting overload sets. The proposed design has been implemented on a fork of the MS-STL [https://github.com/MFHava/STL/tree/P2495](https://github.com/MFHava/STL/tree/P2495).

Proposed Wording
Wording is relative to [N4901]. Additions are presented like this, removals like this.

[version.syn]
In [version.syn], add:

```cpp
#define __cpp_lib_stringview_from_stringview YYYYMM //also in <sstream>
```

Adjust the placeholder value as needed to denote this proposal’s date of adoption.

[stringbuf.general]
In [stringbuf.general], in the synopsis, add the proposed overloads:

```cpp
... // 29.8.2.2, constructors
basic_stringbuf() : basic_stringbuf(ios_base::in | ios_base::out) {}
explicit basic_stringbuf(ios_base::openmode which);
explicit basic_stringbuf(basic_stringbuf<CharT, Traits, Allocator& s, ios_base::openmode which = ios_base::in | ios_base::out>);  
explicit basic_stringbuf(const Allocator& a)  
  basic_stringbuf(ios_base::in | ios_base::out, a) {}  
explicit basic_stringbuf(basic_string<CharT, Traits, Allocator& s, ios_base::openmode which = ios_base::in | ios_base::out>);  
template<class SAlloc>  
basic_stringbuf(basic_string<CharT, Traits, SAlloc& s, const Allocator& a>) : basic_stringbuf(s, ios_base::in | ios_base::out, a) {}  
template<class SAlloc>
```
basic_stringbuf:
  const basic_string<charT, traits, SAlloc>& s,
  ios_base::openmode which, const Allocator& a);
template<class SAlloc>
explicit basic_stringbuf(
  const basic_string<charT, traits, SAlloc>& s,
  ios_base::openmode which = ios_base::in | ios_base::out);
explicit basic_stringbuf(const charT* s);
explicit basic_stringbuf(const Allocator& a);
explicit basic_stringbuf(const basic_stringbuf&); = delete;
basic_stringbuf(basic_stringbuf&&); = delete;
basic_stringbuf(basic_stringbuf& rhs);
basic_stringbuf(basic_stringbuf&& rhs, const Allocator& a);

... // 29.8.2.4, getters and setters
allocator_type get_allocator() const noexcept;

basic_string<charT, traits, Allocator> str() const &;
template<class SAlloc>
basic_string<charT, traits, SAlloc> str(const SAlloc& sa) const;

basic_string<charT, traits, Allocator> str() &&;

basic_string_view<charT, traits> view() const noexcept;
void str(const basic_string<charT, traits, Allocator>& s);
template<class SAlloc>
void str(const basic_string<charT, traits, SAlloc>& s);
void str(basic_string<charT, traits, Allocator>&& s);
void str(const charT* s);
void str(basic_string_view<charT, traits> s);

[stringbuf.cons]
In [stringbuf.cons]:

template<class SAlloc>
explicit basic_stringbuf(
  const basic_string_view<charT, traits, SAlloc>& s,
  ios_base::openmode which = ios_base::in | ios_base::out);

9 Effects: Initializes the base class with basic_streambuf() (29.6.3.2),
mode with which, and buf with s, then calls init_buf_ptrs().

Explicit basic_stringbuf(const charT* s);

9 Effects: Initializes the base class with basic_streambuf() (29.6.3.2),
mode with which, and buf with s, then calls init_buf_ptrs().

Explicit basic_stringbuf(basic_string_view<charT, traits> s);

11 Effects: Initializes the base class with basic_streambuf() (29.6.3.2),
mode with which, and buf with s, then calls init_buf_ptrs().

basic_stringbuf(basic_string_view<charT, traits> s, io_mode which,
  Allocator& a);

11 Effects: Initializes the base class with basic_streambuf() (29.6.3.2),
mode with which, and buf with {s, a}, then calls init_buf_ptrs().

basic_stringbuf(basic_stringbuf& rhs);

[stringbuf.members]
In [stringbuf.members]:

void str(const basic_string<charT, traits, Allocator>& s);
17 Effects: Equivalent to:
  buf = std::move(s);
  init_buf_ptrs();

void str(const charT* s);
18 Effects: Equivalent to:
  buf = s;
  init_buf_ptrs();

void str(basic_string_view<charT, traits> s);
19 Effects: Equivalent to:
  buf = s;
  init_buf_ptrs();
In [istringstream.general], in the synopsis, add the proposed overloads:

```cpp
// 29.8.3.2, constructors
basic_istringstream() : basic_istringstream(ios_base::in) {}
explicit basic_istringstream(ios_base::openmode which);
explicit basic_istringstream(
    const basic_string<charT, traits, Allocator>& s,
    ios_base::openmode which = ios_base::in);
basic_istringstream(ios_base::openmode which, const Allocator& a);
explicit basic_istringstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::in);
template<class SAlloc>
basic_istringstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which, const Allocator& a);
explicit basic_istringstream(
    basic_string_view<charT, traits> s,
    ios_base::openmode which = ios_base::in);

basic_istringstream(basic_istringstream&& rhs);
```

In [istringstream.cons]:

```cpp
explicit basic_istringstream(const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::in);
```

Effects: Initializes the base class with
`basic_istringstream<charT, traits>(addressof(sb))` (29.7.4.2), and `sb` with `basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in)` (29.8.2.2).

```cpp
explicit basic_istringstream(const charT* s,
    ios_base::openmode which = ios_base::in);
```

Effects: Initializes the base class with
`basic_istringstream<charT, traits>(addressof(sb))` (29.7.4.2), and `sb` with `basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in)` (29.8.2.2).

```cpp
explicit basic_istringstream(basic_string_view<charT, traits> s,
    ios_base::openmode which = ios_base::in);
```

Effects: Initializes the base class with
`basic_istringstream<charT, traits>(addressof(sb))` (29.7.4.2), and `sb` with `basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in)` (29.8.2.2).

```cpp
explicit basic_istringstream(basic_istringstream&& rhs);
```
[istringstream.members]
In [istringstream.members]:

```cpp
void str(basic_string<charT, traits, Allocator>&& s);
8 Effects: Equivalent to: rdbuf()->str(std::move(s));

void str(const charT* s);
Effects: Equivalent to: rdbuf()->str(s);

void str(basic_string_view<charT, traits> s);
10 Effects: Equivalent to: rdbuf()->str(s);
```

[ostringstream.general]
In [ostringstream.general], in the synopsis, add the proposed overloads:

```cpp
// 29.8.4.2, constructors
basic_ostreamstream() : basic_ostreamstream(ios_base::out) {} 
explicit basic_ostreamstream(ios_base::openmode which);
explicit basic_ostreamstream(
    const basic_string<charT, traits, Allocator>& s,
    ios_base::openmode which = ios_base::out);
explicit basic_ostreamstream(
    basic_string<charT, traits, Allocator>&& s,
    ios_base::openmode which = ios_base::out);
template<class SAlloc>
basic_ostreamstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::out);
template<class SAlloc>
explicit basic_ostreamstream(
    const basic_string_view<charT, traits> s,
    ios_base::openmode which = ios_base::out);
```

[istringstream.cons]
In [istringstream.cons]:

```cpp
template<class SAlloc>
explicit basic_istreamstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::out);
6 Constraints: is_same_v<SAlloc, Allocator> is false.
7 Effects: Initializes the base class with basic_istream<charT, traits>(addressof(sb)) (29.7.5.2), and sb with basic_string-
buf<charT, traits, Allocator>(s, which | ios_base::out) (29.8.2.2).
```
explicit basic_ostreamstream(const charT* s, 
ios_base::openmode which = ios_base::out);

Effects: Initializes the base class with 
basic_ostream<charT, traits>(addressof(sb)) (29.7.5.2) and 
sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out) (29.8.2).

Explicit basic_ostreamstream(basic_string_view<charT, traits> s, 
ios_base::openmode which = ios_base::out);

Effects: Initializes the base class with 
basic_ostream<charT, traits>(addressof(sb)) (29.7.5.2) and 
sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out) (29.8.2).

basic_ostreamstream(basic_string_view<charT, traits> s, 
ios_base::openmode which,
const Allocator& a);

Effects: Initializes the base class with 
basic_ostream<charT, traits>(addressof(sb)) (29.7.5.2) and 
sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out, a) (29.8.2).

basic_ostreamstream(basic_ostreamstream&& rhs);

[ostringstream.members]
In [ostringstream.members]:

void str(basic_string<charT, traits, Allocator>&& s);
  Effects: Equivalent to: rdbuf() - str(std::move(s));

void str(const charT* s);
  Effects: Equivalent to: rdbuf() - str(s);

void str(basic_string_view<charT, traits> s);
  Effects: Equivalent to: rdbuf() - str(s);

[stringstream.general]
In [stringstream.general], in the synopsis, add the proposed overloads:

... // 29.8.5.2, constructors
basic_stringstream() : basic_stringstream(ios_base::out | ios_base::in) {} // Explicit basic_stringstream(ios_base::openmode which);
explicit basic_stringstream(const basic_string<
charT, traits, Allocator>& s, 
ios_base::openmode which = ios_base::out | ios_base::in);
basic_stringstream(ios_base::openmode which, const Allocator& a);
explicit basic_stringstream(basic_string<charT, traits, Allocator>&& s,
ios_base::openmode which = ios_base::out | ios_base::in);
template<class SAlloc>
basic_stringstream(const basic_string<charT, traits, SAlloc>& s, 
const Allocator& a) : basic_stringstream(s, ios_base::out | ios_base::in, a) {}
template<class SAlloc>
basic_stringstream(
const basic_string<charT, traits, SAlloc>& s, 
ios_base::openmode which, const Allocator& a);
template<class SAlloc>
explicit basic_stringstream(basic_string<charT, traits, SAlloc>&& s,
ios_base::openmode which = ios_base::out | ios_base::in);
explicit basic_stringstream(const basic_string<charT, traits, SAlloc>& s,
ios_base::openmode which = ios_base::out | ios_base::in);
explicit basic_stringstream(basic_string_view<charT, traits> s, 
ios_base::openmode which = ios_base::out | ios_base::in);
basic_stringstream(basic_string_view<charT, traits> s, 
const Allocator& a);
basic_stringstream(const basic_stringstream&) = delete;
basic_stringstream(basic_stringstream&& rhs);
... // 29.8.5.4, members
basic_stringbuf<charT, traits, Allocator>* rdbuf() const;
basic_stringstream<charT, traits, Allocator> str() const &;
template<class SAlloc>
basic_stringstream<charT, traits, SAlloc> str(const SAlloc& sa) const;
basic_stringstream<charT, traits, Allocator> str() &&;
basic_string_view<charT, traits, Allocator> view() const noexcept;
void str(const basic_string<charT, traits, Allocator>&& s);
template<class SAlloc>
void str(const basic_string<charT, traits, SAlloc>& s);
void str(basic_string<charT, traits, Allocator>&& s);
void str(const charT* s);
void str(basic_string_view<charT, traits> s);

[stringstream.cons]
In [stringstream.cons]:

```cpp
template<class SAlloc>
explicit basic_stringstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::out | ios_base::in);
6 Constraints: is_same_v<SAlloc, Allocator> is false.
7 Effects: Initializes the base class with basic_iostream<charT, traits>(addressof(sb)) (29.7.4.7.2), and sb with basic_stringbuf<charT, traits, Allocator>(s, which) (29.8.2.2).

explicit basic_stringstream(const charT* s,
    ios_base::openmode which = ios_base::out | ios_base::in);
8 Effects: Initializes the base class with basic_iostream<charT, traits>(addressof(sb)) (29.7.4.7.2), and sb with basic_stringbuf<charT, traits, Allocator>(s, which) (29.8.2.2).

explicit basic_stringstream(basic_string_view<charT, traits> s,
    ios_base::openmode which = ios_base::out | ios_base::in);
9 Effects: Initializes the base class with basic_iostream<charT, traits>(addressof(sb)) (29.7.4.7.2), and sb with basic_stringbuf<charT, traits, Allocator>(s, which) (29.8.2.2).

basic_stringstream(basic_stringview<charT, traits> s, ios_base::openmode which, const Allocator& a);
10 Effects: Initializes the base class with basic_iostream<charT, traits>(addressof(sb)) (29.7.4.7.2), and sb with basic_stringbuf<charT, traits, Allocator>(s, which, a) (29.8.2.2).

basic_stringbuf(basic_stringbuf&& rhs);
```

[stringstream.members]
In [stringstream.members]:

```cpp
void str(basic_string<charT, traits, Allocator>&& s);
8 Effects: Equivalent to: rdbuf() -> str(std::move(s));

void str(const charT* s);
7 Effects: Equivalent to: rdbuf() -> str(s);

void str(basic_string_view<charT, traits> s);
10 Effects: Equivalent to: rdbuf() -> str(s);
```

Acknowledgements

Thanks to RISC Software GmbH for supporting this work. Thanks to Peter Kulczycki and Bernhard Manfred Gruber for proof reading and discussions.