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.

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>
<tr>
<td>stringstream s15;</td>
<td>stringstream s15;</td>
</tr>
<tr>
<td>s15.str(&quot;&quot;);</td>
<td>s15.str(&quot;&quot;);</td>
</tr>
<tr>
<td>s15.str(&quot;sv&quot;);</td>
<td>s15.str(&quot;sv&quot;);</td>
</tr>
<tr>
<td>s15.str(&quot;s&quot;);</td>
<td>s15.str(&quot;s&quot;);</td>
</tr>
<tr>
<td>s15.str(str);</td>
<td>s15.str(str);</td>
</tr>
<tr>
<td>s15.str(mstr);</td>
<td>s15.str(mstr);</td>
</tr>
<tr>
<td>//concerning LWG2946</td>
<td>//concerning LWG2946</td>
</tr>
<tr>
<td>stringstream s16({&quot;abc&quot;, 1});</td>
<td>stringstream s16({&quot;abc&quot;, 1});</td>
</tr>
<tr>
<td>stringstream s17({&quot;abc&quot;, 1}, alloc);</td>
<td>stringstream s17({&quot;abc&quot;, 1}, alloc);</td>
</tr>
<tr>
<td>stringstream s18({&quot;abc&quot;, 1}, mode, alloc);</td>
<td>stringstream s18({&quot;abc&quot;, 1}, mode, alloc);</td>
</tr>
<tr>
<td>stringstream s19;</td>
<td>stringstream s19;</td>
</tr>
<tr>
<td>s19.str({&quot;abc&quot;, 1});</td>
<td>s19.str({&quot;abc&quot;, 1});</td>
</tr>
</tbody>
</table>

---

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

R0: Initial version

R1: Updates after LEWG Review on 2022-08-16:

- Evaluated [LWG2946](#) based on LEWG feedback.
  - Adjusted proposed design & wording accordingly.
  - Removed evaluation of alternative designs as they are either incompatible with LWG2946 or result in an ABI-break.
  - Dropped support for construction from `const CharT *` with an allocator and an optional openmode.
- Drive-by fix in [stringstream.cons]: added missing Constraints.
- Added section with frequently asked questions.

R2: Updates after LWG Review on 2023-02-10:

- Per LWG guidance merged wording for proposed constructor overloads per class.
- Using `class` instead of `typename` for wording.
- Fixed style of `Effects`-clauses in wording.
- Upgraded referenced standard draft and use stable references in proposed wording.

R3: Updated after LWG Review on 2023-04-05:

- Modified wording style per LWG guidance.
- Removed redundant constraint "is_convertible_v<const T& , const CharT *> is false" (originally taken from LWG2946). This change enables constructions with `const CharT *`, an allocator, and an optional openmode.

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|o]stringstream 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/reini-
itialization from a string, the potential design is presented in terms of CLASS:

```cpp
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&);

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

    template<typename SAlloc>
    requires(!std::is_same_v<SAlloc, Alloc>)
    explicit CLASS(basic_string<CharT, Traits, SAlloc>&&);

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

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>Moving the string, used for const CharT *.</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>Copying the string.</td>
</tr>
<tr>
<td>6</td>
<td>Equal to 5 but input has different allocator. Invalid for const CharT *.</td>
</tr>
<tr>
<td>7</td>
<td>Moving the string, used for const CharT *.</td>
</tr>
</tbody>
</table>

We propose to add restricted basic_string_view-overloads for 1, 2, 3, 6:

```cpp
template<typename T>
static
constexpr
bool is_string_view_like_v(std::is_convertible_v<const T &, basic_string_view<CharT, Traits>>); //exposition only

//add to existing class definition:
template<typename T>
requires is_string_view_like_v
explicit CLASS(const T&, ios_base::openmode = /*def*/);

template<typename T>
requires is_string_view_like_v
CLASS(const T&, const Alloc&);

template<typename T>
requires is_string_view_like_v
CLASS(const T&, ios_base::openmode, const Alloc&);

template<typename T>
requires is_string_view_like_v
void str(const T&);
```
Impact on the Standard
This proposal is a pure library addition. Existing standard library classes are modified in a non-ABI-breaking way. Overload resolution for singular const CharT * arguments changes from constructing a temporary string to constructing a string_view.

Implementation Experience
The proposed overload set has been implemented on [https://godbolt.org/z/T9P73P9sP] for evaluation. Additionally, the proposed design has been implemented on a fork of the MS-STL [https://github.com/MFHava/STL/tree/P2495].

Frequently Asked Questions
Why is this needed when C++23 includes spanstream?
 Whilst there certainly is an overlap between basic_spanstream and basic_stringstream, fundamental differences in their semantics (ownership & growability) preclude the former to be a drop-in replacement for all conceivable uses of the latter.

Proposed Wording
Wording is relative to [N4944]. Additions are presented like this, removals like this and drafting notes like this.

[version.syn]

```
#define __cpp_lib_stringstream_from_string_view YYYYMM //also in <sstream>

[DRAFTING NOTE: Adjust the placeholder value as needed to denote this proposal’s date of adoption.]
```

### [stringbuf]

```cpp
31.8.2 Class template basic_stringbuf
31.8.2.1 General
namespace std {
    template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
    class basic_stringbuf : public basic_streambuf<charT, traits> {
        // [stringbuf.cons], constructors
        template<class SAlloc>
        explicit basic_stringbuf(
            const basic_string<charT, traits, SAlloc>& s,
            ios_base::openmode which = ios_base::in | ios_base::out);

        template<class T>
        basic_stringbuf(const T& t, ios_base::openmode which, const Allocator& a);

        template<class T>
        basic_stringbuf(const T& t, const Allocator& a);

        template<class T>
        basic_stringbuf(const basic_stringbuf&); // [stringbuf.members], getters and setters

        protected:
            void str(basic_string<charT, traits, Allocator>&& s);
    }
}
```

Additional evaluation of all overload sets presented in R0 can be found here: [https://godbolt.org/z/esWWr6hTr]
Effects: Initializes the base class with \( \text{basic\_streambuf}() \) (\[streambuf.cons\]), mode with which, and buf with s, then calls \( \text{init\_buf\_ptrs}() \).

\[
\begin{align*}
\text{template\_class S} & : \text{basic\_stringbuf}(\text{const } \& \text{t}, \text{ios\_base\_openmode which} = \text{ios\_base\_in} | \text{ios\_base\_out}) \\
\text{template\_class S} & : \text{basic\_stringbuf}(\text{const } \& \text{t}, \text{const Allocator } \& \text{a}) \\
\text{template\_class S} & : \text{basic\_stringbuf}(\text{const } \& \text{t}, \text{ios\_base\_openmode which}, \text{const Allocator } \& \text{a}) \\
\text{let which be } \text{ios\_base\_in} | \text{ios\_base\_out} \text{ for the overload with no parameter which, and } \text{a be Allocator()} \text{ for the overload with no parameter } \text{a}.
\end{align*}
\]

- \( \text{basic\_stringbuf}(\text{basic\_stringbuf} & \text{rhs}); \)
- \( \text{basic\_stringbuf}(\text{basic\_stringbuf} & \text{rhs}, \text{const Allocator } \& \text{a}); \)

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.2.4 Member functions

- \( \text{void str(\text{basic\_string<charT}, traits, Allocator } \& \text{& s}); } \)

31.8.2.5 Overridden virtual functions

### [istringstream]

31.8.3 Class template \texttt{basic\_istringstream}

31.8.3.1 General

\[
\begin{align*}
\text{namespace std} & \{
\text{template<class charT, class traits = char\_traits<charT>, class Allocator = allocator<charT>}>
\text{class basic\_istringstream : public basic\_istream<charT, traits> \{} & \\
\text{\hspace{1cm} \text{\text{- // [istringstream.cons], constructors}} & \\
\text{\hspace{1cm} \text{\text{- template\_class SAlloc}} & \\
\text{\hspace{1cm} \hspace{1cm} explicit basic\_istringstream(} & \\
\text{\hspace{1cm} \hspace{1cm} \text{const basic\_string<charT, traits, SAlloc } \& \text{& s,}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{ios\_base\:_openmode which = ios\_base\:_in);}} & \\
\text{\hspace{1cm} \text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} explicit basic\_istringstream(} & \\
\text{\hspace{1cm} \hspace{1cm} \text{const basic\_string<charT, traits, Allocator } \& \text{a)};}} & \\
\text{\hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} basic\_istringstream(} & \\
\text{\hspace{1cm} \hspace{1cm} \text{const basic\_istringstream&);}} & \\
\text{\hspace{1cm} \text{- // [istringstream.members], members}} & \\
\text{\hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} str(\text{basic\_string<charT, traits, Allocator } \& \text{& s);}} & \\
\text{\hspace{1cm} \text{- \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} str(} & \\
\text{\hspace{1cm} \hspace{1cm} \text{const charT &);}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{private:}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{- });}} & \\
\text{\text{\}}} & \\
\end{align*}
\]

31.8.3.2 Constructors

\[
\begin{align*}
\text{template\_class SAlloc} & : \text{basic\_istringstream(} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- \text{DRAFTING NOTE: Drive-by fix, this adds a missing constraint present in streamstring and ostrstream.)}}}} & \\
\text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- \text{DRAFTING NOTE: Drive-by fix, this adds a missing constraint present in streamstring and ostrstream.)}}}} & \\
\text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- \text{DRAFTING NOTE: Drive-by fix, this adds a missing constraint present in streamstring and ostrstream.)}}}} & \\
\text{\text{- template\_class S}} & \\
\text{\hspace{1cm} \hspace{1cm} \text{\text{- \text{DRAFTING NOTE: Drive-by fix, this adds a missing constraint present in streamstring and ostrstream.)}}}} \}
\end{align*}
\]
```cpp
basic_istringstream(basic_istringstream&& rhs);

31.8.3.4 Member functions

```
31.8.5 Class template `basic_stringstream`

31.8.5.1 General

namespace std {
    template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
    class basic_stringstream : public basic_ios<charT, traits> {
        // [stringstream.cons], constructors
        template<class SAlloc>
        explicit basic_stringstream(
            const basic_string<charT, traits, SAlloc>& s,
            ios_base::openmode which = ios_base::out | ios_base::in);
        template<class T>
        explicit basic_stringstream(const T& t, ios_base::openmode which = ios_base::out | ios_base::in);
        template<class T>
        basic_stringstream(const T& t, const Allocator& a);
        template<class T>
        basic_stringstream(const T& t, ios_base::openmode which, const Allocator& a);
        basic_stringstream(const basic_stringstream&) = delete;
        // [stringstream.members], members
        private:
    };
}

31.8.5.2 Constructors

template<class SAlloc>
explicit basic_stringstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::out | ios_base::in);
Constraints: is_same_v<SAlloc, Allocator> is false.
Effects: Initializes the base class with basic_ios<charT, traits>::openmode which, and sb with basic_stringbuf<charT, traits, Allocator>(s, which) [stringbuf.cons].

template<class T>
explicit basic_stringstream(const T& t, ios_base::openmode which = ios_base::out | ios_base::in);
template<class T>
basic_stringstream(const T& t, const Allocator& a);
template<class T>
basic_stringstream(const T& t, ios_base::openmode which, const Allocator& a);
Constraints: is_convertible_v<const T&, basic_string_view<charT, traits>> is true.
Effects: Initializes the base class with addressof(sb) and direct-list-initializes sb with t, which, a.

basic_stringstream(basic_stringstream&& rhs);

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.5.4 Member functions

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

void str(const T& t);
Constraints: is_convertible_v<const T&, basic_string_view<charT, traits>> is true.
Effects: Equivalent to: rdbuf() -> str(t);

Acknowledgements

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