Document number:	P2495R0
Date:	2022-02-10
Project:	Programming Language C++
Audience:	LEWG
Reply-to:	Michael Florian Hava[1] <mfh.cpp@gmail.com>

Interfacing stringstreams 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

Before		Proposed	
`const ios_base::openmode mode;`		`const ios_base::openmode mode;`	
`const allocator<char> alloc;`		`const allocator<char> alloc;`	
`const string str;`		`const string str;`	
`//implicitly convertible to string_view`		`//implicitly convertible to string_view`	
`const mystring mstr;`		`const mystring mstr;`	
`stringstream s0{""};`	✓	`stringstream s0{""};`	✓
`stringstream s1{"", alloc};`	✗	`stringstream s1{"", alloc};`	✓
`stringstream s2{"", mode, alloc};`	✗	`stringstream s2{"", mode, alloc};`	✓
`stringstream s3{""sv};`	✗	`stringstream s3{""sv};`	✓
`stringstream s4{""sv, alloc};`	✗	`stringstream s4{""sv, alloc};`	✓
`stringstream s5{""sv, mode, alloc};`	✗	`stringstream s5{""sv, mode, alloc};`	✓
`stringstream s6{""s};`	✓	`stringstream s6{""s};`	✓
`stringstream s7{""s, alloc};`	✓	`stringstream s7{""s, alloc};`	✓
`stringstream s8{""s, mode, alloc};`	✓	`stringstream s8{""s, mode, alloc};`	✓
`stringstream s9{str};`	✓	`stringstream s9{str};`	✓
`stringstream s10{str, alloc};`	✓	`stringstream s10{str, alloc};`	✓
`stringstream s11{str, mode, alloc};`	✓	`stringstream s11{str, mode, alloc};`	✓
`stringstream s12{mstr};`	✗	`stringstream s12{mstr};`	✓
`stringstream s13{mstr, alloc};`	✗	`stringstream s13{mstr, alloc};`	✓
`stringstream s14{mstr, mode, alloc};`	✗	`stringstream s14{mstr, mode, alloc};`	✓
`stringstream s15;`		`stringstream s15;`	
`s15.str("");`	✓	`s15.str("");`	✓
`s15.str(""sv);`	✗	`s15.str(""sv);`	✓
`s15.str(""s);`	✓	`s15.str(""s);`	✓
`s15.str(str);`	✓	`s15.str(str);`	✓
`s15.str(mstr);`	✗	`s15.str(mstr);`	✓

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|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/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*/);          1

  template<typename SAlloc>
  CLASS(const basic_string<CharT, Traits, SAlloc>&, const Alloc&);                                   2

  template<typename SAlloc>
  CLASS(const basic_string<CharT, Traits, SAlloc>&, ios_base::openmode, const Alloc&);               3

  template<typename SAlloc>
  requires(!std::is_same_v<Alloc, SAlloc>)
  explicit CLASS(const basic_string<CharT, Traits, SAlloc>&, ios_base::openmode = /*def*/);         4

  explicit CLASS(basic_string<CharT, Traits, Alloc>&&, ios_base::openmode = /*def*/);                5


  //reinitialization of internal string
  void str(const basic_string<CharT, Traits, Alloc>&);                                               6

  template<typename SAlloc>
  requires(!std::is_same_v<Alloc, SAlloc>)
  void str(const basic_string<CharT, Traits, SAlloc>&);                                              7

  void str(basic_string<CharT, Traits, Alloc>&&);                                                    8
```

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

No	Description
1	Copying the string.
2	Copying the string, input may have different allocator. Invalid for `const CharT *`.
3	
4	Equal to 1 but input has different allocator. Invalid for `const CharT *`.
5	Moving the string, used for `const CharT *`.
6	Copying the string.
7	Equal to 6 but input has different allocator. Invalid for `const CharT *`.
8	Moving the string, used for `const CharT *`.

There are several possible designs to add support for `basic_string_view`, some of which also fix the unsupported construction from `const CharT *` in $\boxed{2}$ $\boxed{3}$.

Option 1: Add additional overloads

Introduce `basic_string_view`-overloads for $\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{6}$. The overloads to $\boxed{1}$ $\boxed{6}$ 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 $\boxed{1}$ $\boxed{6}$ – overload resolution for existing usages of `const CharT *` changes from $\boxed{5}$ $\boxed{8}$ to the new semantically equivalent overloads.

```
//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 $\boxed{1}$ $\boxed{6}$; handling of a single `const CharT *` is unchanged. Parameters implicitly convertible to `basic_string_view` are **not** supported by this approach!

```
//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 $\boxed{1}$ $\boxed{5}$ $\boxed{6}$ $\boxed{8}$; `const CharT *` is handled by the overloads of $\boxed{1}$ $\boxed{6}$. Depending on the implementation strategy, this might result in an ABI break.

```
//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 $\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{6}$ to `basic_string_view`; remove $\boxed{4}$ $\boxed{7}$ as they are no longer needed; restrict $\boxed{5}$ $\boxed{8}$ to disambiguate handling of `const CharT *` with $\boxed{1}$ $\boxed{6}$. These extensive changes result in an ABI break.

```
//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] 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].

Proposed Wording

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

[version.syn]

In [version.syn], add:

```
#define __cpp_lib_sstream_from_string_view YYYYMML //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:

```
...
    // 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(
      const basic_string<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) {}
    basic_stringbuf(ios_base::openmode which, const Allocator& 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(
        const 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,
      ios_base::openmode which = ios_base::in | ios_base::out);
    explicit basic_stringbuf(basic_string_view<charT, traits> s,
      ios_base::openmode which = ios_base::in | ios_base::out);
    basic_stringbuf(basic_string_view<charT, traits> s, const Allocator& a)
      : basic_stringbuf(s, ios_base::in | ios_base::out, a) {}
    basic_stringbuf(basic_string_view<charT, traits> s, ios_base::openmode which,
      const Allocator& a);
    basic_stringbuf(const 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<charT, traits, SAlloc>& s,
    ios_base::openmode which = ios_base::in | ios_base::out);
```

8 *Constraints*: `is_same_v<SAlloc, Allocator>` is `false`.

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,
  ios_base::openmode which = ios_base::in | ios_base::out);
```

10 *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,
  ios_base::openmode which = ios_base::in | ios_base::out);
```

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, ios_base::openmode which,
  const Allocator& a);
```

12 *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(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();
```

[istringstream.general]

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

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

...
    // 29.8.3.4, members
    basic_stringbuf<charT, traits, Allocator>* rdbuf() const;

    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);
```

[istringstream.cons]

In [istringstream.cons]:

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

6 *Effects*: Initializes the base class with `basic_istream<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).

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

7 *Effects*: Initializes the base class with `basic_istream<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).

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

8 *Effects*: Initializes the base class with `basic_istream<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).

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

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

```
basic_istringstream(basic_istringstream&& rhs);
```

[istringstream.members]

In [istringstream.members]:

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

```
void str(const charT* s);
```
9 *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:

```
...
    // 29.8.4.2, constructors
    basic_ostringstream() : basic_ostringstream(ios_base::out) {}
    explicit basic_ostringstream(ios_base::openmode which);
    explicit basic_ostringstream(
      const basic_string<charT, traits, Allocator>& s,
      ios_base::openmode which = ios_base::out);
    basic_ostringstream(ios_base::openmode which, const Allocator& a);
    explicit basic_ostringstream(
      basic_string<charT, traits, Allocator>&& s,
      ios_base::openmode which = ios_base::out);
    template<class SAlloc>
      basic_ostringstream(
        const basic_string<charT, traits, SAlloc>& s, const Allocator& a)
        : basic_ostringstream(s, ios_base::out, a) {}
    template<class SAlloc>
      basic_ostringstream(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which, const Allocator& a);
    template<class SAlloc>
      explicit basic_ostringstream(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which = ios_base::out);
    explicit basic_ostringstream(const charT* s,
      ios_base::openmode which = ios_base::out);
    explicit basic_ostringstream(basic_string_view<charT, traits> s,
      ios_base::openmode which = ios_base::out);
    basic_ostringstream(basic_string_view<charT, traits> s, const Allocator& a)
      : basic_ostringstream(s, ios_base::out, a) {}
    basic_ostringstream(basic_string_view<charT, traits> s, ios_base::openmode which,
      const Allocator& a);
    basic_ostringstream(const basic_ostringstream&) = delete;
    basic_ostringstream(basic_ostringstream&& rhs);

...
    // 29.8.4.4, members
    basic_stringbuf<charT, traits, Allocator>* rdbuf() const;

    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);
```

[ostringstream.cons]

In [ostringstream.cons]:

```
template<class SAlloc>
  explicit basic_ostringstream(
    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_ostream<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_ostringstream(const charT* s,
  ios_base::openmode which = ios_base::out);
```
8 *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)` (29.8.2.2).

```
explicit basic_ostringstream(basic_string_view<charT, traits> s,
  ios_base::openmode which = ios_base::out);
```
9 *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)` (29.8.2.2).

```
basic_ostringstream(basic_string_view<charT, traits> s, ios_base::openmode which,
  const Allocator& a);
```
10 *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)` (29.8.2.2).

```
basic_ostringstream(basic_ostringstream&& rhs);
```

[ostringstream.members]

In [ostringstream.members]:

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

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

```
void str(basic_string_view<charT, traits> s);
```
10 *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(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which = ios_base::out | ios_base::in);
    explicit basic_stringstream(const charT* 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(s, ios_base::out | ios_base::in, a) {}
    basic_stringstream(basic_string_view<charT, traits> s, ios_base::openmode which,
      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_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);
```

[stringstream.cons]

In [stringstream.cons]:

```
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_string-buf<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_string-buf<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_string-buf<charT, traits, Allocator>(s, which)` (29.8.2.2).

```
basic_stringstream(basic_string_view<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_string-buf<charT, traits, Allocator>(s, which, a)` (29.8.2.2).

```
basic_stringbuf(basic_stringbuf&& rhs);
```

[stringstream.members]

In [stringstream.members]:

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

```
void str(const charT* s);
```
9 *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.