Document number:	P2495R2
Date:	2023-02-14
Project:	Programming Language C++
Audience:	LWG
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`.

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);`	✓
`//concerning LWG2946`		`//concerning LWG2946`	
`stringstream s16({"abc", 1});`	✓	`stringstream s16({"abc", 1});`	✓
`stringstream s17({"abc", 1}, alloc);`	✗	`stringstream s17({"abc", 1}, alloc);`	✗
`stringstream s18({"abc", 1}, mode, alloc);`	✗	`stringstream s18({"abc", 1}, mode, alloc);`	✗
`stringstream s19;`		`stringstream s19;`	
`s19.str({"abc", 1});`	✓	`s19.str({"abc", 1});`	✓

[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 [istringstream.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.

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 3	Copying the string, input may have different allocator. Invalid for `const CharT *`.
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 *`.

I propose to add restricted `basic_string_view`-overloads for 1 2 3 6:

```
template<typename T>
static
constexpr
bool is_string_view_like_v{std::is_convertible_v<const T&, std::basic_string_view<CharT, Traits>> &&
                           !std::is_convertible_v<const T&, const CharT*>}; //exposition only


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

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

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

template<typename T>
requires is_string_view_like_v<T>
void str(const T&);
```

Due to following the design of LWG2946, constructions with `const CharT *`, an allocator, and an optional openmode (akin to 2 3) remains unsupported.

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 existing code is not affected by the introduced overloads.

Implementation Experience

The proposed overload set has been implemented on [https://godbolt.org/z/vo5c5P6eT] for evaluation[2]. 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 [N4928]. Additions are presented like this, removals like ~~this~~ and drafting notes like **this**.

[version.syn]

```
#define __cpp_lib_sstream_from_string_view YYYYMML //also in <sstream>
```

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

[stringbuf]

31.8.2 Class template basic_stringbuf **[stringbuf]**

31.8.2.1 General **[stringbuf.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>
      explicit basic_stringbuf(const T& t, ios_base::openmode which = ios_base::in | ios_base::out);
    template<class T>
      basic_stringbuf(const T& t, const Allocator& a);
    template<class T>
      basic_stringbuf(const T& t, ios_base::openmode which, const Allocator& a);
    basic_stringbuf(const basic_stringbuf&) = delete;
…
    // [stringbuf.members], getters and setters
…
    void str(basic_string<charT, traits, Allocator>&& s);
    template<class T>
      void str(const T& t);
  protected:
…
  };
}
```

31.8.2.2 Constructors **[stringbuf.cons]**

```
…
template<class SAlloc>
```

[2] An updated evaluation of all overload sets presented in R0 can be found here: https://godbolt.org/z/esWWr6hTr

```
  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()` (**[*streambuf.cons*]**), `mode` with `which`, and `buf` with `s`, then calls `init_buf_ptrs()`.

```
template<class T>
  explicit basic_stringbuf(const T& t, ios_base::openmode which = ios_base::in | ios_base::out);
template<class T>
  basic_stringbuf(const T& t, const Allocator& a);
template<class T>
  basic_stringbuf(const T& t, ios_base::openmode which, const Allocator& a);
```

10 Let `which` be `ios_base::in | ios_base::out` for the overload that does not accept `which` as a parameter, and `a` be `Allocator()` for the overload that does not accept `a` as a parameter.

11 *Constraints:*

(11.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(11.2) — `is_convertible_v<const T&, const charT*>` is `false`.

12 *Effects:* Initializes the base class with `basic_streambuf()` (**[*streambuf.cons*]**), `mode` with `which`, and `buf` with `{t,a}`, then calls `init_buf_ptrs()`.

```
basic_stringbuf(basic_stringbuf&& rhs);
basic_stringbuf(basic_stringbuf&& rhs, const Allocator& a);
```

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.2.4 Member functions **[stringbuf.members]**

…

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

17 *Effects*: Equivalent to:

```
  buf = std::move(s);
  init_buf_ptrs();
```

```
template<class T>
  void str(const T& t);
```

18 *Constraints:*

(18.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(18.2) — `is_convertible_v<const T&, const charT*>` is `false`.

19 *Effects:* Equivalent to:

```
  buf = t;
  init_buf_ptrs();
```

31.8.2.5 Overridden virtual functions **[stringbuf.virtuals]**

[istringstream]

31.8.3 Class template `basic_istringstream` **[istringstream]**

31.8.3.1 General **[istringstream.general]**

```
namespace std {
  template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
  class basic_istringstream : public basic_istream<charT, traits> {
…
    // [istringstream.cons], constructors
…
    template<class SAlloc>
      explicit basic_istringstream(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which = ios_base::in);
    template<class T>
      explicit basic_istringstream(const T& t, ios_base::openmode which = ios_base::in);
    template<class T>
      basic_istringstream(const T& t, const Allocator& a);
    template<class T>
      basic_istringstream(const T& t, ios_base::openmode which, const Allocator& a);
    basic_istringstream(const basic_istringstream&) = delete;
…
    // [istringstream.members], members
…
    void str(basic_string<charT, traits, Allocator>&& s);
    template<class T>
      void str(const T& t);
  private:
…
  };
}
```

31.8.3.2 Constructors **[istringstream.cons]**

…

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

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

[DRAFTING NOTE: Drive-by fix, this adds a missing constraint present in stringstream and ostringstream.]

~~6~~7 *Effects*: Initializes the base class with `basic_istream<charT, traits>(addressof(sb))` (*[istream]*), and `sb` with `basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in)` (*[stringbuf.cons]*).

```
template<class T>
  explicit basic_istringstream(const T& t, ios_base::openmode which = ios_base::in);
template<class T>
  basic_istringstream(const T& t, const Allocator& a);
template<class T>
  basic_istringstream(const T& t, ios_base::openmode which, const Allocator& a);
```

8 Let `which` be `ios_base::in` for the overload that does not accept `which` as a parameter, and `a` be `Allocator()` for the overload that does not accept `a` as a parameter.

9 *Constraints*:

(9.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(9.2) — `is_convertible_v<const T&, const charT*>` is `false`.

10 *Effects*: Initializes the base class with `basic_istream<charT, traits>(addressof(sb))` (*[istream]*) and `sb` with `basic_stringbuf<charT, traits, Allocator>(t, which | ios_base::in, a)` (*[stringbuf.cons]*).

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

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.3.4 Member functions [istringstream.members]

…

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

8 *Effects*: Equivalent to: `rdbuf()->str(std::move(s));`

```
template<class T>
  void str(const T& t);
```

9 *Constraints*:

(9.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(9.2) — `is_convertible_v<const T&, const charT*>` is `false`.

10 *Effects*: Equivalent to `rdbuf()->str(t)`.

[ostringstream]

31.8.4 Class template `basic_ostringstream` [ostringstream]

31.8.4.1 General [ostringstream.general]

```
namespace std {
  template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
  class basic_ostringstream : public basic_ostream<charT, traits> {
…
    // [ostringstream.cons], constructors
…
    template<class SAlloc>
      explicit basic_ostringstream(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which = ios_base::out);
    template<class T>
      explicit basic_ostringstream(const T& t, ios_base::openmode which = ios_base::out);
    template<class T>
      basic_ostringstream(const T& t, const Allocator& a);
    template<class T>
      basic_ostringstream(const T& t, ios_base::openmode which, const Allocator& a);
    basic_ostringstream(const basic_ostringstream&) = delete;
…
    // [ostringstream.members], members
…
    void str(basic_string<charT, traits, Allocator>&& s);
    template<class T>
      void str(const T& t);
  private:
…
  };
}
```

31.8.4.2 Constructors [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))` (*[ostream]*), and `sb` with `basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out)` (*[stringbuf.cons]*).

```
template<class T>
  explicit basic_ostringstream(const T& t, ios_base::openmode which = ios_base::out);
template<class T>
  basic_ostringstream(const T& t, const Allocator& a);
template<class T>
  basic_ostringstream(const T& t, ios_base::openmode which, const Allocator& a);
```

8 Let `which` be `ios_base::out` for the overload that does not accept `which` as a parameter, and `a` be `Allocator()` for the overload that does not accept `a` as a parameter.

9 *Constraints*:

(9.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(9.2) — is_convertible_v<const T&, const charT*> is false.

10 *Effects*: Initializes the base class with basic_ostream<charT, traits>(addressof(sb)) ([*ostream*]) and sb with basic_stringbuf<charT, traits, Allocator>(t, which | ios_base::out, a) ([*stringbuf.cons*]).

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

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.4.4 Member functions **[ostringstream.members]**

…

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

8 *Effects*: Equivalent to: rdbuf()->str(std::move(s));

```
template<class T>
  void str(const T& t);
```

9 *Constraints*:

(9.1) — is_convertible_v<const T&, basic_string_view<charT, traits>> is true, and

(9.2) — is_convertible_v<const T&, const charT*> is false.

10 *Effects*: Equivalent to rdbuf()->str(t).

[stringstream]

31.8.5 Class template basic_stringstream **[stringstream]**

31.8.5.1 General **[stringstream.general]**

```
namespace std {
  template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
  class basic_stringstream : public basic_iostream<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
…
    void str(basic_string<charT, traits, Allocator>&& s);
    template<class T>
      void str(const T& t);
  private:
…
  }
};
```

31.8.5.2 Constructors **[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)) ([*iostream.cons*]), 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);
```

8 Let which be ios_base::out | ios_base::in for the overload that does not accept which as a parameter, and a be Allocator() for the overload that does not accept a as a parameter.

9 *Constraints*:

(9.1) — is_convertible_v<const T&, basic_string_view<charT, traits>> is true, and

(9.2) — is_convertible_v<const T&, const charT*> is false.

10 *Effects*: Initializes the base class with basic_iostream<charT, traits>(addressof(sb)) ([*iostream.cons*]) and sb with basic_stringbuf<charT, traits, Allocator>(t, which, a) ([*stringbuf.cons*]).

```
basic_stringstream(basic_stringstream&& rhs);
```

[DRAFTING NOTE: Renumber remaining constructors.]

31.8.5.4 Member functions **[stringstream.members]**

…

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

8 *Effects*: Equivalent to: rdbuf()->str(std::move(s));

```
template<class T>
```

```
  void str(const T& t);
```

9 *Constraints*:

(9.1) — `is_convertible_v<const T&, basic_string_view<charT, traits>>` is `true`, and

(9.2) — `is_convertible_v<const T&, const charT*>` is `false`.

10 *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.