1 Motivation

Streams have been the oldest part of the C++ standard library and their specification doesn’t take into account many things introduced since C++11. One of the oversights is that there is no non-copying access to the internal buffer of a `basic_stringbuf` which makes at least the obtaining of the output results from an `ostringstream` inefficient, because a copy is always made. I personally speculate that this was also the reason why `basic_strbuf` took so long to get deprecated with its `char *` access.

With move semantics and `basic_string_view` there is no longer a reason to keep this pessimisation alive on `basic_stringbuf`.

I also believe we should remove `basic_strbuf` from the standard’s appendix [depr.str.strstreams]. This is proposed in p0448, that completes the replacement of that deprecated feature.

2 Introduction

This paper proposes to adjust the API of `basic_stringbuf` and the corresponding stream class templates to allow accessing the underlying string more efficiently.

C++17 and library TS have `basic_string_view` allowing an efficient read-only access to a contiguous sequence of characters which I believe `basic_stringbuf` has to guarantee about its internal buffer, even if it is not implemented using `basic_string` obtaining a `basic_string_view` on the internal buffer should work sidestepping the copy overhead of calling `str()`.

On the other hand, there is no means to construct a `basic_string` and move from it into a `basic_stringbuf` via a constructor or a move-enabled overload of `str(basic_string &&)`. 
2.1 History

2.1.1 Changes from r5

There was a review in my absence (again) in San Diego, November 2018. I’ll try my best to incorporate the feedback here.

— rebase on n4791.

— undo premature application of p1163 (explicit -> non-explicit multi-parameter ctors by additional overloads) (ARGHHH, but I now think I follow Titus argumentation that it might be a bad idea).

— see table 1. LEWG might need to reconsider the combination of p0407/p0408 to agree on sane ctor overloads. LWG and Ville gave feedback on different ctor overloads. New Design: separate SFINAEd overloads for "foreign allocator" string arguments.

— clean up \texttt{str()} member function overloads. This was in the overlap of p0407 and p0408 and not seen by LEWG in that way (sorry!). Split getter to two, one taking an allocator for the new string. Setter \texttt{str(string const&)} member function remains a template on the string’s Allocator.

— adjust the \textit{italic} explanations accordingly to the changes.

— LWG question: Do the constructors taling a \texttt{SAlloc} template parameter restrict it to \texttt{Cpp17Allocator} requirements? It is implicit via \texttt{basic_string}.

— Fixed a specification bug in move construction allowing keeping the original wording of move-assign. \texttt{rhs} must be "synced" first, by relying initializing \texttt{buf} from \texttt{std::move(rhs).str()} instead from \texttt{std::move(rhs.buf)} directly.

— merge getters \texttt{str()} specification of \texttt{high_mark} into a single specification for simplification and consistency.

— split copying setters \texttt{str(basic_string const &)} into the previously existing one and the one taking a \texttt{basic_string} with a different allocator, like with the constructors to reduce ABI problems.

— drive by editorial fix to mention already existing Allocator template parameter in stream classes, i.e., \texttt{basic__sstream<charT, traits, Allocator>} where mentioned in descriptions

— drive-by fix to postcondition of \texttt{basic_stringbuf} move constructor to also refer to \texttt{getloc()}.

2.1.2 Changes from r4

Incorporate suggestions from LWG review in Batavia, August 2018. This was the first time the combined proposal was reviewed.

— Adjust specification sections to new naming schema introduced at the Rapperswil Meeting 2018 for C++20. (Requires->(Mandates (compile-time), Expects(contract)), Remarks->Constraints, Postconditions->Ensures).

— change the overloads of constructors with default arguments to only have the single argument version explicit according to p1163.

— introduce an exposition-only member function \texttt{init_buf_ptrs()} in \texttt{basic_stringbuf} to set
Table 1 — Overview of stringbuf/stringstream constructors

<table>
<thead>
<tr>
<th>string</th>
<th>which</th>
<th>Allocator</th>
<th>ctor</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
<td>default</td>
<td></td>
<td>exists</td>
</tr>
<tr>
<td>copy</td>
<td>opt</td>
<td>explicit</td>
<td></td>
<td>exists</td>
</tr>
<tr>
<td>move</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>407 for stateful allocs</td>
</tr>
<tr>
<td>other-copy</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>407 other kind of strings</td>
</tr>
<tr>
<td>other-copy</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>407 other kind of strings</td>
</tr>
<tr>
<td>other-copy</td>
<td>opt</td>
<td>explicit</td>
<td></td>
<td>407 above with default alloc</td>
</tr>
<tr>
<td>move</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>408r5 - useless, copies anyway</td>
</tr>
</tbody>
</table>

1 other-copy means has a different Allocator template argument
2 allow if same or different allocator for string, because allocator is given, see copy-ctor string with allocator parameter.
3 LEWG new design question: requires string Allocator different from stringbuf Allocator (new), otherwise existing ctor is changed ABI (and may be CTAD) breakage.
* in addition a move ctor is defined taking an additional Allocator argument like with basic_string
** Allocator should always be the last Parameter (is that really always the case?).

the streambuf pointers. In the standard version, that was part of the str(string) member function and now is needed in more than one place. Add a note there about internally violating invariants of buf. Explain the exposition only members in the front matter of the class.

— reduce clutter, since bit operations are possible in enum ios_base::openmode parameters.

— Provide a note that allocator properties are propagated along the properties of the basic_string member buf in the front matter of the class. I hope this is sufficient to address the issue from Batavia about what happens with the allocators. Also all other allocator relevant comments should be addressed through that delegation to basic_string’s properties.

— I provided the following definition of swap for basic_stringbuf adopted from basic_string. Note, the base class swap does not give a noexcept guarantee. I provided that:

```cpp
void swap(basic_stringbuf& s)
    noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value || allocator_traits<Allocator>::is_always_equal::value);
```

— basic_stringbuf move constructor now guarantees that rhs is empty, as if std::move(rhs).str() was called. This seems the easiest way to guarantee its get and put area are re-initialized accordingly. While technically not required, it makes handling moved-from streambufs (which are rare) consistent with calling the rvalue-ref-qualified str() member function. Please note, we do not give such a guarantee to the moved-from state of the stream objects, since they
get their buffer pointer stolen and thus are completely unrelated to a streambuf after been moved-from. Only moving out the underlying string keeps the stream in working condition.

### 2.1.3 Changes from r3
To make the job of reviewing and integrating my stringstream adjustments easier, I incorporate the changes proposed in p0407r2 (allocator-aware basic_stringbuf), since both papers have been forwarded by LEWG to LWG.

- Added full set of reasonable overloads to the constructors with and without allocator (basic_string& does not get an allocator constructor template argument to allow efficient construction from charT* literals).

### 2.1.4 Changes from r2
Discussed in Albuquerque, where LEWG was in favor to forward it to LWG for IS with the following change.

- reestablish rvalue-ref qualified str() instead of the previously suggested pilfer().
- address LWG only in document header.

### 2.1.5 Changes from r1
Discussed in LEWG Issaquah. Answering some questions and raising more. Reflected in this paper.

- reflected new section numbers from the std. now relative to the current working draft.
- implementation is now working with gcc 7. (not relevant for this paper)

### 2.1.6 Changes from r0

- Added more context to synopsis sections to see all overloads (Thanks Alisdair).
- rename str_view() to just view(). There was discussion on including an explicit conversion operator as well, but I didn’t add it yet (my implementation has it).
- renamed r-value-ref qualified str() to pilfer() and removed the reference qualification from it and remaining str() member.
- Added allocator parameter for the basic_string parameter/result to member functions (see p0407 for allocator support for stringstreams in general)

### 3 Acknowledgements

- Daniel Krügler encouraged me to pursue this track.
- Alisdair Meredith for telling me to include context in the synopsis showing all overloads. That is the only change in this version, no semantic changes!
- Jonathan Wakely to show me the #undef _GLIBCXX_EXTERN_TEMPLATE
4 Impact on the Standard

This is an extension to the API of basic_stringbuf, basic_stringstream, basic_istreamstream, and basic_ostringstream class templates.

This paper addresses both Library Fundamentals TS 3 and C++ Next (2020?). When added to the standard draft with p0448 (spanstream), section [depr.str.streams] should be removed.

5 Design Decisions

After experimentation I decided that substituting the \((\text{basic\_string}<\text{charT,traits,Allocator const \\&})\) constructors in favor of passing a \(\text{basic\_string\_view}\) would lead to ambiguities with the new move-from-string constructors.

5.1 Hint to implementers

In both libc++ and libstdc++ I needed to make basic_stringbuf a friend of basic_string to allow efficient growth of the buffer beyond the current string length (breaking an invariant) until it is retrieved using one of the str() member functions. Other implementations might use a different strategy of caring for the buffer space that should be efficiently be adopted by the returned string object, thus requiring either special basic_string constructors or access to its internals as well.

5.2 Open Issues to be discussed by LWG

Note: this list includes the discussion of p0407 features.

- Does it make sense to add \text{nolexcept} specifications for \text{move()} and \text{swap()} members, since the base classes and other streams do not. At least it does not make sense so for stream objects, since the base classes do not specify that.

- The basic_string constructors that move from the string get a default template argument for \SAlloc in the hope that allows initialization from a character string literal. Need confirmation that this trick works and selects the better constructor for temporary conversion without ambiguity, because for the copying (const-ref) overload the allocator of the string needs to be deduced. This should lead to the effect of optimizing existing usages.

5.3 Open Issues discussed by LEWG in Albuquerque

- Should pilfer() be rvalue-ref qualified to denote the "destruction" of the underlying buffer? LEWG in Issaquah didn’t think so, but I’d like to ask again. LEWG small group in Albuquerque in favor of rvalue-ref qualification. \text{Re-establish str()&}, drop pilfer

5.4 Open Issues discussed by LEWG in Issaquah and Albuquerque

- Is the name of the str_view() member function ok? No. Renamed to view()

- Should the str()& overload be provided for move-out? \text{No. give it another name (pilfer)} and remove rvalue-ref qualification (Issaquah). \text{Re-establish str()&}, drop pilfer
— Should `str()` && empty the character sequence or leave it in an unspecified but valid state? Empty it, and specify.
— Provide guidance on validity lifetime of of the obtained `string_view` object.

### 5.5 Open Issues to be discussed by LEWG/LWG (in Kona?)

— LEWG: Please look at constructor overloads (see Table 1) and `str()` overloads that came from the mix of p0407 with p0408.
— Both: Constructor overloads taking a string with a different allocator, esp. SFINAE. Is that OK?
— LWG: Is `!is_same_v<SAlloc, Allocator>` the correct SFINAE predicate for foreign allocator overloads?
— LEWG: `noexcept` for `view()` member function of stringbuf (note streams have a precondition on this call and can not be noexcept)(suggested by LWG).
— LWG: Does an Allocator template parameter that is mapped to `basic_string`’s Allocator template parameter need to conform to `Cpp17Allocator` requirements or is that given implicit by its usage?
— LWG: recheck wording.

### 6 Technical Specifications

The following is relative to n4791.
Remove section on `char*` streams [depr.str.strstreams] and all its subsections from appendix D.

#### 6.1 28.8.2 Adjust synopsis of basic_stringbuf [stringbuf]

Add a new constructor overload.

`Note that p0407 provides allocator support for basic_stringbuf, since now both papers have been forwarded to LWG, the changes proposed in p0407 are integrated here for ease of review and integration. The explanations of those changes are added in italics here. from r6 on some changes that need to be revisited by LEWG are made, since the overlap of the two papers’ functionality.

Change each of the non-moving, non-deleted constructors to add a const-ref `Allocator` parameter as last parameter with a default constructed `Allocator` as default argument. Add an overload for the move constructor adding an `Allocator` parameter like with `basic_string`. Add an exposition-only member variable `buf` to allow referring to it for specifying allocator behaviour. May be: Add noexcept specification, depending on allocator behavior, like with `basic_string`?

This section also adopts the changes of p1163 by only marking the single argument constructors explicit and provide non-explicit overloads for zero, two or more argument versions. That paper p1163 was tentatively accepted in Batavia, August 2018.

```cpp
// [stringbuf.cons], 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>& str,
  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>& str,
  const Allocator& a)
  : basic_stringbuf(str, ios_base::in | ios_base::out, a) {}

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>& str,
  ios_base::openmode which,
  const Allocator& a);
template<class SAlloc>
basic_stringbuf(
  const basic_string<charT, traits, SAlloc>& str,
  const Allocator& a) : basic_stringbuf(str, ios_base::in | ios_base::out, a) {}

explicit basic_stringbuf(
  const basic_string<charT, traits, SAlloc>& str,
  ios_base::openmode which = ios_base::in | ios_base::out);

basic_stringbuf(const basic_stringbuf& rhs) = delete;
basic_stringbuf(basic_stringbuf&& rhs);
basic_stringbuf(basic_stringbuf& rhs, const Allocator& a);

The following list summarizes the edits:

- Add an rvalue-ref overload of \texttt{str()} that obtains the underlying string via moving from \texttt{buf}.
- Add a \texttt{str(Allocator)} overload template member function to take an \texttt{Allocator} for the returned string and add a reference qualification the existing \texttt{str()} overload. NEW: was intermingled with existing \texttt{str()} member, now separate.
- Add the \texttt{view()} member function obtaining a \texttt{string_view} to the underlying internal buffer. NEW: make that noexcept.
- Add a setter \texttt{str()} overload as a template member function copying into the string buffer to take an allocator template parameter that differs from the buffer's own \texttt{Allocator}.
- Add a \texttt{str(string&&)} overload that moves from its string rvalue-reference argument into the internal buffer.
- Provide an exposition-only member function \texttt{init_buf_ptrs()} to ensure streambuf pointers are initialized correctly by all \texttt{buf} setting operations.

// [stringbuf.members], \texttt{getters} and \texttt{setters}:
basic_string<charT, traits, Allocator> str() const &;

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

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);

Add the following declaration to the public section of synopsis of the class template basic_stringbuf:

allocator_type get_allocator() const noexcept;

Add the following exposition only member to the private section of synopsis of the class template basic_stringbuf. This allows to delegate all details of allocator-related behaviour on what basic_string is doing, simplifying this specification a lot.

private:
    ios_base::openmode mode; // exposition only
    basic_string<charT, traits, Allocator> buf; // exposition only
    void init_buf_ptrs(); // exposition only

Adjust p2 of the section to include the additionbal exposition only members and add a note on the allocator properties of basic_stringbuf.

1 The class basic_stringbuf is derived from basic_streambuf to associate possibly the input sequence and possibly the output sequence with a sequence of arbitrary characters. The sequence can be initialized from, or made available as, an object of class basic_string.

2 For the sake of exposition, the maintained data and internal pointer initialization is presented here as:

(2.1) — ios_base::openmode mode, has in set if the input sequence can be read, and out set if the output sequence can be written.

(2.2) — basic_string<charT, traits, Allocator> buf contains the underlying character sequence.

(2.3) — init_buf_ptrs() sets the base class’ get area ([streambuf.get.area]) and put area ([streambuf.put.area]) pointers after initializing, moving from, or assigning to buf accordingly.

6.1.1 28.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Adjust the constructor specifications taking the additional Allocator parameter and an overload for the move-constructor taking an Allocator. Make the constructors’ wording that actually construct a buf consistent.

explicit basic_stringbuf(ios_base::openmode which);

1 Effects: Constructs an object of class basic_stringbuf, initializing the base class with basic_streambuf() [streambuf.cons], and initializing mode with which. It is implementation-defined whether the sequence pointers (eback(), gptr(), egptr(), pbase(), pptr(), eptr()) are initialized to null pointers.
```
Ensures: str() == "."

explicit basic_stringbuf(
    const basic_string<CharT, traits, Allocator>& s,
    ios_base::openmode which = ios_base::in | ios_base::out);

Effects: Constructs an object of class basic_stringbuf, initializing the base class with
basic_streambuf() [streambuf.cons], and initializing mode with which, and buf with s. Calls
init_buf_ptrs(). Then calls str(s).

basic_stringbuf(
    ios_base::openmode which,
    const Allocator &a);

Effects: Constructs an object of class basic_stringbuf, initializing the base class with
basic_stringbuf() [streambuf.cons], initializing mode with which, and buf with a. Calls
init_buf_ptrs().

Ensures: str() == "."

explicit basic_stringbuf(
    basic_string<CharT, traits, Allocator>&& s,
    ios_base::openmode which = ios_base::in | ios_base::out);

Effects: Constructs an object of class basic_stringbuf, initializing the base class with
basic_stringbuf() [streambuf.cons], initializing mode with which, and buf with std::move(s). Calls
init_buf_ptrs().

template<class SAlloc>
basic_stringbuf(
    basic_string<CharT, traits, SAlloc>& s,
    ios_base::openmode which,
    const Allocator &a);

Effects: Constructs an object of class basic_stringbuf, initializing the base class with
basic_stringbuf() [streambuf.cons], initializing mode with which, and buf with {s,a}. Calls
init_buf_ptrs().

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

Constraints: !is_same_v<SAlloc, Allocator>.

Effects: Constructs an object of class basic_stringbuf, initializing the base class with
basic_stringbuf() [streambuf.cons], initializing mode with which, and buf with s. Calls
init_buf_ptrs().

Add the additional move constructor taking an allocator and adjust the description accordingly:

basic_stringbuf(basic_stringbuf&& rhs);

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

Note to LWG reviewers: using std::move(rhs).str() ensures rhs.buf is in a consistent state before the
move happens. Before the spec was wrong, because rhs.buf might have been shorter than the actual written characters. Also a drive by (IMHO editorial fix) better spelling out what happens since we now have the exposition only members.

Effects: Move constructs from the rvalue rhs. This is accomplished by copy constructing the base class and initializing mode with rhs.mode. In the first form buf is initialized from `std::move(rhs).str()`. In the second form buf is initialized from `{std::move(rhs).str(), a}`. It is implementation-defined whether the sequence pointers in *this (eback(), gptr(), egptr(), pbase(), pptr(), epptr()) obtain the values which rhs had. Whether they do or not, *this and rhs reference separate buffers (if any at all) after the construction. The openmode, locale and any other state of rhs is also copied.

Ensures: Let rhs_p refer to the state of rhs just prior to this construction and let rhs_a refer to the state of rhs just after this construction.

\begin{align}
\text{— } \text{str()} & = \text{rhs_p.str()} \\
\text{— } \text{gptr() - eback()} & = \text{rhs_p.gptr()} - \text{rhs_p.eback()} \\
\text{— } \text{egptr() - eback()} & = \text{rhs_p.egptr()} - \text{rhs_p.eback()} \\
\text{— } \text{pptr() - pbase()} & = \text{rhs_p.pptr()} - \text{rhs_p.pbase()} \\
\text{— } \text{epptr() - pbase()} & = \text{rhs_p.epptr()} - \text{rhs_p.pbase()} \\
\text{— } \text{if (eback()) eback()} & \neq \text{rhs_a.eback()} \\
\text{— } \text{if (gptr()) gptr()} & \neq \text{rhs_a.gptr()} \\
\text{— } \text{if (egptr()) egptr()} & \neq \text{rhs_a.egptr()} \\
\text{— } \text{if (pbase()) pbase()} & \neq \text{rhs_a.pbase()} \\
\text{— } \text{if (pptr()) pptr()} & \neq \text{rhs_a.pptr()} \\
\text{— } \text{if (epptr()) epptr()} & \neq \text{rhs_a.epptr()} \\
\text{— } \text{getloc()} & = \text{rhs_p.getloc()} \\
\text{— } \text{rhs is empty but usable, as if std::move(rhs).str() was called.}
\end{align}

### 6.2 28.8.2.2 Assign and swap [stringbuf.assign]

Most of this section is included to allow for simpler adding of conditional noexcept.

basic_stringbuf& operator=(basic_stringbuf&& rhs);

Effects: After that move assignment *this has the observable state it would have had if it had been move constructed from rhs (see [stringbuf.cons]).

Returns: *this.

void swap(basic_stringbuf& rhs);

\begin{verbatim}
noexcept(allo...::propagate_on_container_swap::value || allocator_traits<Alloc...>::is_always_equal::value);
\end{verbatim}

Effects: Exchanges the state of *this and rhs.

template <class charT, class traits, class Allocator>
void swap(basic_stringbuf<charT, traits, Allocator>& x,
        basic_stringbuf<charT, traits, Allocator>& y);

Effects: As if by x.swap(y).

6.2.1 28.8.2.3 Member functions [stringbuf.members]

Provide a section introducing paragraph explaining the high-water-mark. The wording is taken directly
from n4791 [stringbuf.members] p.1 with some grammar adjustment to adjust to the fact that we
now have multiple setters. Introduce the exposition only private member functions init_buf_ptrs() to provide the correct intialization of streambuf pointer members and adjust the str(s) member
functions with parameters accordingly.

1 The member functions getting the underlying character sequence all refer to a high_mark value,
where high_mark represents the position one past the highest initialized character in the buffer.
Characters can be initialized by writing to the stream, by constructing the basic_stringbuf with a
basic_string, or by calling one of the str(basic_string) setter member functions. In the case
of calling one of the str(basic_string) setter member functions, all characters initialized prior
to the call are now considered uninitialized (except for those characters re-initialized by the new
basic_string).

void init_buf_ptrs(); // exposition only

Effects: Initializes the input and output sequences from buf according to mode.

Ensures: If mode & ios_base::out is true, pbase() points to buf.front() and
eptr() >= pbase() + buf.size() holds; in addition, if mode & ios_base::ate is true,
pptr() == pbase() + buf.size() holds, otherwise pptr() == pbase() is true.
If mode & ios_base::in is true, eback() points to buf.front(), and both
gptr() == eback() and egptr() == eback() + buf.size() hold.

[Note: For efficiency reasons stream buffer operations might violate invariants of buf while
it is held encapsulated in the basic_stringbuf, i.e., by writing to characters in the range
[buf.data()+buf.size(), buf.data()+buf.capacity()]. All operations retrieving a basic_string
from buf ensure that the basic_string invariants hold on the returned value. Implementations might make basic_stringbuf a friend class to basic_string to achieve that.
—end note]

Add the definition of the get_allocator function:

allocator_type get_allocator() const noexcept;

Returns: buf.get_allocator().

Add a getter overload taking an allocator parameter for the copied from string to allow having a
different allocator than the underlying stream and add a ref-qualifier to the existing getter overload
to avoid ambiguities with the rvalue-ref qualified overload. Add a getter overload that is rref qualified
and mention it.

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

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

Change p1 (now p2) to use plural for "str(basic_string) member functions" and refer to the allocator. Constraints on SAlloc were taken from [container.requirements.general] p.17:

Returns: A basic_string object whose content is equal to the basic_stringbuf’s underlying character sequence. In the second form the basic_string<charT, traits, SAlloc> is constructed with allocator sa. If the basic_stringbuf was created only in input mode, the resultant basic_string contains the character sequence in the range [eback(), egptr()). If the basic_stringbuf was created with which & ios_base::out being true then the resultant basic_string contains the character sequence in the range [pbase(), high_mark), where high_mark represents the position one past the highest initialized character in the buffer. Characters can be initialized by writing to the stream, by constructing the basic_stringbuf with a basic_string, or by calling the str(basic_string) member function. In the case of calling the str(basic_string) member function, all characters initialized prior to the call are now considered uninitialized (except for those characters re-initialized by the new basic_string). Otherwise the basic_stringbuf has been created in neither input nor output mode and a zero length basic_string is returned.

Constraints:

(7.1) — The qualified-id SAlloc::value_type is valid and denotes a type[temp.deduct].
(7.2) — The expression declval<SAlloc&>().allocate(size_t{}) is well-formed when treated as an unevaluated operand.

Add the following specifications for str()&& and view() const member function:

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

Returns: A basic_string<charT, traits, Allocator> object move constructed from the basic_stringbuf’s underlying character sequence in buf. This can be achieved by first adjusting buf to the underlying character sequence as follows.

If the basic_stringbuf was created only in input mode, adjust buf to contain basic_string(eback(), egptr()-eback()). If the basic_stringbuf was created with which & ios_base::out being true then adjust buf to contain basic_string(pbase(), high_mark-pbase()). Otherwise the basic_stringbuf has been created in neither input nor output mode and a zero length basic_string is returned.

[Note: The explained effect can be achieved by setting buf’s size() to the resulting length (egptr()–eback() or high_mark–pbase()) and returning std::move(buf). However, basic_string does not provide setting the length without changing the characters in the buffer in the range [buf.data()+buf.size(), buf.data()+buf.capacity()) as a public member function. — end note]

Ensures: The underlying character sequence buf is empty and pbase(), ppstr(), eback(), gptr(), egptr() are initialized as if calling init_buf_ptrs() with an empty buf.

basic_string_view<charT, traits> view() const;

Returns: A basic_string_view object referring to the basic_stringbuf’s underlying character sequence in buf. If the basic_stringbuf was created only in input mode, basic_string_view::
view(eback(), egptr()-eback()). If the basic_stringbuf was created with which & ios_base::out being true then basic_string_view(pbase(), high_mark-pbase())). Otherwise the basic_stringbuf has been created in neither input nor output mode and a basic_string_view referring to an empty range is returned.

[Note: Using the returned basic_string_view object after destruction or any modification of the character sequence underlying *this, such as output on the holding stream, will cause undefined behavior, because the internal string referred to by the return value might have changed or been re-allocated. — end note]

add setter overloads and simplify their specification through relying on buf and init_buf_ptrs().

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

Effects: buf = std::move(s); init_buf_ptrs().

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

Effects: buf = s; init_buf_ptrs(). Copies the content of s into the basic_stringbuf underlying character sequence and initializes the input and output sequences according to mode.

Ensures: If mode & ios_base::out is nonzero, pbase() points to the first underlying character and egptr() >= pbase() + s.size() holds; in addition, if mode & ios_base::ate is nonzero, pptr() == pbase() + s.size() holds, otherwise pptr() == pbase() is true. If mode & ios_base::in is nonzero, eback() points to the first underlying character, and both gptr() == eback() and egptr() == eback() + s.size() hold.

6.3 28.8.3 Adjust synopsis of basic_istringstream [iostream.stream]

Provide constructor overloads taking an Allocator argument and also those that allow a string with a different allocator type.

// [iostream.stream.cons], constructors:
basic_istringstream() : basic_istringstream(ios_base::in) {}
explicit basic_istringstream(ios_base::openmode which);
explicit basic_istringstream(
    const basic_string<charT, traits, Allocator>& str,
    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);

basic_istringstream(const basic_istringstream& rhs) = delete;
basic_istringstream(basic_istringstream&& rhs);

Adjust getter/setter members according to basic_stringbuf:

// [istringstream.members], 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_view<charT, Traits> view() const ;

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);

6.3.1 28.8.3.1 basic_istringstream constructors [istringstream.cons]

Adjust the constructor specifications analog to basic_stringbuf. Deliberately do not provide the special
move constructor taking an allocator. Drive-by editorial fix to include Allocator template argument.

explicit basic_istringstream(ios_base::openmode which);

1 Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>,
initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(which | ios_base::in)
[streambuf.cons].

explicit basic_istringstream(
    const basic_string<charT, traits, Allocator>& str,
    ios_base::openmode which = ios_base::in);

2 Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>,
initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::in)
[streambuf.cons].
const Allocator& a);

Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>, initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream] and initializing sb with basic_stringbuf<charT, traits, Allocator>(which | ios_base::in, a) ([stringbuf.cons]).

explicit basic_istringstream(
    basic_string<charT, traits, Allocator>&& s,
    ios_base::openmode which = ios_base::in);

Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>, initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream] and initializing sb with basic_stringbuf<charT, traits, Allocator>(std::move(s), which | ios_base::in) [stringbuf.cons].

template<class SAlloc>
    basic_istringstream(
        const basic_string<charT, traits, SAlloc>& s,
        ios_base::openmode which,
        const Allocator& a);

Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>, initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream] and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in, a) ([stringbuf.cons]).

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

Constraints: !is_same_v<SAlloc, Allocator>.

Effects: Constructs an object of class basic_istringstream<charT, traits, Allocator>, initializing the base class with basic_istream<charT, traits>(addressof(sb))[istream] and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::in) [stringbuf.cons]).

6.3.2  28.8.3.3 Member functions [istringstream.members]

Extend str() overloads according to basic_stringbuf and add view():

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

Returns: rdbuf()->str().

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

Returns: rdbuf()->str(sa).

Constraints:

(3.1)  The qualified-id SAlloc::value_type is valid and denotes a type[temp.deduct].
The expression `declval<SAloc&>().allocate(size_t{})` is well-formed when treated as an unevaluated operand.

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

Returns: `std::move(*rdbuf()).str()`.

```cpp
basic_string_view<charT, traits> view() const;
```

Returns: `rdbuf()->view()`.

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

Effects: Calls `rdbuf()->str(s)`.

```cpp
template<class SAloc>
void str(const basic_string<charT, traits, SAloc>& s);
```

Constraints: `!is_same_v<SAloc,Allocator>`.

Effects: Calls `rdbuf()->str(s)`.

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

Effects: Calls `rdbuf()->str(std::move(s))`.

### 6.4 28.8.4 Adjust synopsis of basic_ostringstream [ostringstream]

Provide constructor overloads taking an Allocator argument and also those that allow a string with a different allocator type.

```cpp
// [ostringstream.cons], constructors:
basic_ostringstream() : basic_ostringstream(ios_base::in) {}
explicit basic_ostringstream(ios_base::openmode which);
explicit basic_ostringstream(
    const basic_string<charT, traits, Allocator>& str,
    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 SAloc>
basic_ostringstream(
    const basic_string<charT, traits, SAloc>& s,
    const Allocator& a) : basic_ostringstream(s, ios_base::out, a) {}
```

```cpp
template <class SAloc>
basic_ostringstream(
    const basic_string<charT, traits, SAloc>& s,
    ios_base::openmode which ,
    const Allocator& a);
```
const basic_string<charT, traits, SAlloc>& s,
ios_base::openmode which = ios_base::out);

basic_ostringstream(const basic_ostringstream& rhs) = delete;
basic_ostringstream(basic_ostringstream&& rhs);

Adjust getter/setter members according to basic_stringbuf:

Adjust the constructor specifications analog to basic_stringbuf. deliberately do not provide the special
move constructor taking an allocator. Drive-by editorial fix to include Allocator template argument.

explicit basic_ostringstream(ios_base::openmode which);

Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(which | ios_base::out)
[stringbuf.cons].

explicit basic_ostringstream(
    const basic_string<charT, traits, Allocator>& str,
    ios_base::openmode which = ios_base::out);

Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::out)
[stringbuf.cons].

basic_ostringstream(
    ios_base::openmode which,
    const Allocator& a);

Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(which | ios_base::out, a)
[stringbuf.cons].

explicit basic_ostringstream(

basic_string<charT, traits, Allocator>&& s,
ios_base::openmode which = ios_base::out);

Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(std::move(s), which
| ios_base::out) [stringbuf.cons].

4 template<class SAlloc>
  basic_ostringstream(
      const basic_string<charT, traits, SAlloc>& s,
      ios_base::openmode which,
      const Allocator& a);

Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out, a) [stringbuf.cons]).

5 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>.

7 Effects: Constructs an object of class basic_ostringstream<charT, traits, Allocator>,
initializing the base class with basic_ostream<charT, traits>(addressof(sb))[ostream]
and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which | ios_base::out) [stringbuf.cons]).

6.4.2 28.8.4.3 Member functions [ostringstream.members]
Extend str() overloads according to basic_stringbuf and add view():

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

1 Returns: rdbuf()->str().

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

2 Returns: rdbuf()->str(sa).

3 Constraints:
(3.1) — The qualified-id SAlloc::value_type is valid and denotes a type[temp.deduct].
(3.2) — The expression declval<SAlloc&>().allocate(size_t{}) is well-formed when treated
as an unevaluated operand.

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

4 Returns: std::move(*rdbuf()).str().

basic_string_view<charT, traits> view() const;
Returns: rdbuf()->view().

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

Effects: Calls rdbuf()->str(s).

template<class SAlloc>
void str(const basic_string<charT, traits, SAlloc>& s);

Constraints: !is_same_v<SAlloc, Allocator>.

Effects: Calls rdbuf()->str(s).

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

Effects: Calls rdbuf()->str(std::move(s)).

6.5 28.8.5 Adjust synopsis of basic_stringstream [stringstream]

Provide constructor overloads taking an Allocator argument and also those that allow a string with a
different allocator type.

// [stringstream.cons], 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>& str,
    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);

basic_stringstream(const basic_stringstream& rhs) = delete;
basic_stringstream(basic_stringstream&& rhs);

Adjust getter/setter members according to basic_stringbuf:

// [ostringstream.members], members:
6.5.1 28.8.4.1 basic_stringstream constructors [stringstream.cons]

Adjust the constructor specifications analog to basic_stringbuf. Deliberately do not provide the special move constructor taking an allocator. Drive-by editorial fix to include Allocator template argument.

explicit basic_stringstream(ios_base::openmode which);

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb)) [iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(which) [stringbuf.cons].

explicit basic_stringstream(
    const basic_string<charT, traits, Allocator>& str,
    ios_base::openmode which = ios_base::out | ios_base::in);

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb)) [iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(str, which) [stringbuf.cons].

basic_stringstream(
    ios_base::openmode which,
    const Allocator& a);

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb)) [iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(which, a) [stringbuf.cons].

explicit basic_stringstream(
    basic_string<charT, traits, Allocator>&& s,
    ios_base::openmode which = ios_base::out | ios_base::in);

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb)) [iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(std::move(s), which) [stringbuf.cons].

template<class SAlloc>
basic_stringstream(
    const basic_string<charT, traits, SAlloc>& s,
    ios_base::openmode which,
    const Allocator& a);

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb))[iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which, a)[[stringbuf.cons]].

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>.

Effects: Constructs an object of class basic_stringstream<charT, traits, Allocator>, initializing the base class with basic_iostream<charT, traits>(addressof(sb))[iostream.cons] and initializing sb with basic_stringbuf<charT, traits, Allocator>(s, which)[[stringbuf.cons]].

6.5.2 28.8.4.3 Member functions [stringstream.members]

Extend str() overloads according to basic_stringbuf and add view():

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

Returns: rdbuf()->str().

template<class SAlloc>

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

Returns: rdbuf()->str(sa).

Constraints:

(3.1) — The qualified-id SAlloc::value_type is valid and denotes a type[temp.deduct].

(3.2) — The expression declval<SAlloc&>().allocate(size_t{}) is well-formed when treated as an unevaluated operand.

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

Returns: std::move(*rdbuf()).str().

basic_string_view<charT, traits> view() const;

Returns: rdbuf()->view().

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

Effects: Calls rdbuf()->str(s).

template<class SAlloc>

void str(const basic_string<charT, traits, SAlloc>& s);
Constraints: !is_same_v<SAlloc, Allocator>.

Effects: Calls rdbuf()->str(s).

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

Effects: Calls rdbuf()->str(std::move(s)).

## Appendix: Example Implementations

The given specification has been implemented within a recent version of the sstream header of gcc8. Modified version of the headers and some tests are available at [https://github.com/PeterSommerlad/SC22WG21_Papers/tree/master/workspace/Test_basic_stringbuf_efficient/src](https://github.com/PeterSommerlad/SC22WG21_Papers/tree/master/workspace/Test_basic_stringbuf_efficient/src).

A corresponding implementation for clang 7 is available in the vicinity of the one above at: [https://github.com/PeterSommerlad/SC22WG21_Papers/tree/master/workspace/Test_clang_p0407_p0408](https://github.com/PeterSommerlad/SC22WG21_Papers/tree/master/workspace/Test_clang_p0407_p0408).