

p0408r3 - Efficient Access to `basic_stringbuf`'s Buffer

Peter Sommerlad

2017-11-16

| | |
|------------------|--------------------------|
| Document Number: | p0408r3 |
| Date: | 2017-11-16 |
| Project: | Programming Language C++ |
| Audience: | LWG |
| Target: | C++20 |

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 `ostreamstream` 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 \[depr.str.strstreams\].](#)

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 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.

2.1.2 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)

2.1.3 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_istringstream`, 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.strstreams] should be removed.

5 Design Decisions

After experimentation I decided that substituting the `(basic_string<charT,traits,Allocator const &)` constructors in favor of passing a `basic_string_view` would lead to ambiguities with the new move-from-string constructors.

5.1 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. [Re-establish `str\(\)&&`, drop `pilfer`](#)

5.2 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? ~~No. give it another name (`pilfer`) and remove rvalue-ref qualification (Issaquah).~~ [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.

6 Technical Specifications

The following is relative to n4604.

Remove section on `char*` streams [depr.str.strstreams] and all its subsections from appendix D.

6.1 30.8.2 Adjust synopsis of `basic_stringbuf` [stringbuf]

Add a new constructor overload.

Note that p0407 provides allocator support for `basic_stringbuf`, depending on acceptance sequence of the papers, some overloads need to be further added/adjusted and the wording also adjusted. At the time of p0408r3 that was forwarded to LWG, p0407 was not yet reviewed by LEWG.

```
// ??, constructors:
explicit basic_stringbuf(
    ios_base::openmode which = ios_base::in | ios_base::out);
template<class SAlloc = Allocator>
explicit basic_stringbuf(
    const basic_string<charT, traits, SAllocator>& str,
    ios_base::openmode which = ios_base::in | ios_base::out);
explicit basic_stringbuf(
    basic_string<charT, traits, Allocator>&& s,
    ios_base::openmode which = ios_base::in | ios_base::out);
```

```
basic_stringbuf(const basic_stringbuf& rhs) = delete;
basic_stringbuf(basic_stringbuf&& rhs);
```

Change the getting `str()` overload to take an `Allocator` for the returned string and add a reference qualification. Add an rvalue-ref overload of `str()`. Change the `str()` overload copying into the string buffer to take an allocator template parameter that could differ from the buffer's own `Allocator`. Add a `str()` overload that moves from its string rvalue-reference argument into the internal buffer. Add the `view()` member function obtaining a `string_view` to the underlying internal buffer.

```
// ??, get and set:
template<class SAlloc = Allocator>
basic_string<charT, traits, SAllocator> str(const SAlloc& sa = SAlloc()) const &;
template<class SAlloc = Allocator>
void str(const basic_string<charT, traits, SAllocator>& s);

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

6.1.1 30.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Modify the following constructor specification:

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

- ¹ *Effects:* Constructs an object of class `basic_stringbuf`, initializing the base class with `basic_streambuf()` (??), and initializing mode with `which`. Then calls `str(s)`.

Add the following constructor specification:

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

- ² *Effects:* Constructs an object of class `basic_stringbuf`, initializing the base class with `basic_streambuf()` (30.6.3.1), and initializing mode with `which`. Then calls `str(std::move(s))`.

Note to editors: if p0407 is accepted the changes there for allocators apply here as well. However, different allocators for `s` and the `basic_stringbuf` will result in a copy instead of a move.

6.1.2 30.8.2.3 Member functions [stringbuf.members]

Add an allocator parameter for the copied from string to allow having a different allocator than the underlying stream and a ref-qualifier to avoid ambiguities with the rvalue-ref qualified overload.

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

Change p1 to use plural for "str(basic_string) member functions" and refer to the allocator:

- ¹ *Returns:* A `basic_string` object with allocator `sa` whose content is equal to the `basic_stringbuf` underlying character sequence. 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 one of the `str(basic_string)` member functions. In the case of calling one of the `str(basic_string)` member functions, 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.

Add the following specifications and adjust the wording of `str() const` according to the wording given for `view() const` member function.:

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

2 *Effects:* Moves the content of `s` into the `basic_stringbuf` underlying character sequence and initializes the input and output sequences according to `mode`.

3 *Postconditions:* Let `size` denote the original value of `s.size()` before the move. If `mode & ios_base::out` is true, `pbase()` points to the first underlying character and `egptr() >= pbase() + size` holds; in addition, if `mode & ios_base::ate` is true, `pptr() == pbase() + size` holds, otherwise `pptr() == pbase()` is true. If `mode & ios_base::in` is true, `eback()` points to the first underlying character, and both `gptr() == eback()` and `egptr() == eback() + size` hold.

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

4 *Returns:* A `basic_string` object moved from the `basic_stringbuf` underlying character sequence. If the `basic_stringbuf` was created only in input mode, `basic_string(eback(), egptr()-eback())`. If the `basic_stringbuf` was created with `which & ios_base::out` being true then `basic_string(pbase(), high_mark-pbase())`, 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)` member functions. In the case of calling one of the `str(basic_string)` member functions, 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 an empty `basic_string` is returned.

5 *Postconditions:* The underlying character sequence is empty.

6 [*Note:* After calling this member function the `basic_stringbuf` object remains usable. — *end note*]

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

7 *Returns:* A `basic_string_view` object referring to the `basic_stringbuf` underlying character sequence. If the `basic_stringbuf` was created only in input mode, `basic_string_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())`, 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)` member functions. In the case of calling one of the `str(basic_string)` member functions, 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 `basic_string_view` referring to an empty range is returned.

8 [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 by the return value might have changed or re-allocated. — end note]

6.2 30.8.3 Adjust synopsis of `basic_istream` [`istream`]

Add a new constructor overload and change the one taking the string by copy to allow a different allocator for the copied from string:

```
// ??, constructors:
explicit basic_istream(
    ios_base::openmode which = ios_base::in);
template<class SAlloc = Allocator>
explicit basic_istream(
    const basic_string<charT, traits, SAllocator>& str,
    ios_base::openmode which = ios_base::in);

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

basic_istream(const basic_istream& rhs) = delete;
basic_istream(basic_istream&& rhs);
```

Change the getting `str()` overload to take an `Allocator` for the returned string and add a reference qualification. Add an rvalue-ref overload of `str()`. Change the `str(s)` overload to take an allocator template parameter that could differ from the buffer's own `Allocator`. Add a `str(s)` overload that moves from its string and a `view()` member function:

```
// ??, members:
basic_stringbuf<charT, traits, Allocator>* rdbuf() const;

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

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

6.2.1 30.8.3.1 `basic_istream` constructors [`istream.cons`]

Change the constructor specification to allow a string copy with a different allocator.

```

template<class SAlloc = Allocator>
explicit basic_istream(
    const basic_string<charT, traits, SAllocator>& str,
    ios_base::openmode which = ios_base::in);

```

- 1 *Effects:* Constructs an object of class `basic_istream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::in)` (??).

Add the following constructor specification:

```

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

```

- 2 *Effects:* Constructs an object of class `basic_istream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(str), which | ios_base::in)` (30.8.2.1).

6.2.2 30.8.3.3 Member functions [istream.members]

Add the allocator parameter to the following `str()` overloads:

```

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

```

- 1 *Returns:* `rdbuf()->str(sa)`.

```

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

```

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

Add the following specifications:

```

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

```

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

```

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

```

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

- 5 [*Note:* Calling this member function leaves the stream object in a usable state with an emptied underlying `basic_stringbuf`. — *end note*]

```

basic_string_view<charT, traits> view() const;

```

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

6.3 30.8.4 Adjust synopsis of `basic_ostringstream` [ostringstream]

Add a new constructor overload and change the one taking the string by copy to allow a different allocator for the copied from string:

```

// ??, constructors:
explicit basic_ostringstream(

```

```

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

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

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

```

Change the getting `str()` overload to take an `Allocator` for the returned string and add a reference qualification. Add an rvalue-ref overload of `str()`. Change the `str(s)` overload to take an allocator template parameter that could differ from the buffer's own `Allocator`. Add a `str(s)` overload that moves from its string and a `view()` member function:

```

// ??, members:
basic_stringbuf<charT, traits, Allocator>* rdbuf() const;

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

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

```

6.3.1 30.8.4.1 basic_ostringstream constructors [ostreamstream.cons]

Change the constructor specification to allow a string copy with a different allocator.

```

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

```

- ¹ *Effects:* Constructs an object of class `basic_ostringstream<charT, traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::out)` (??).

Add the following constructor specification:

```

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

```

- ² *Effects:* Constructs an object of class `basic_ostringstream<charT, traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(str), which | ios_base::out)` (30.8.2.1).

6.3.2 30.8.4.3 Member functions [ostreamstream.members]

Add the allocator parameter to the following `str()` overloads:

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

1 *Returns:* `rdbuf()->str(sa)`.

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

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

Add the following specifications:

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

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

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

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

5 [*Note:* Calling this member function leaves the stream object in a usable state with an emptied underlying `basic_stringbuf`. — *end note*]

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

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

6.4 30.8.5 Adjust synopsis of `basic_stringstream` [stringstream]

Add a new constructor overload and change the one taking the string by copy to allow a different allocator for the copied from string:

```
// ??, constructors:
explicit basic_stringstream(
    ios_base::openmode which = ios_base::out | ios_base::in);
template<class SAlloc=Allocator>
explicit basic_stringstream(
    const basic_string<charT, traits, SAllocator>& str,
    ios_base::openmode which = ios_base::out | ios_base::in);
explicit basic_stringstream(
    basic_string<charT, traits, Allocator>&& str,
    ios_base::openmode which = ios_base::in | ios_base::out);
basic_stringstream(const basic_stringstream& rhs) = delete;
basic_stringstream(basic_stringstream&& rhs);
```

Change the getting `str()` overload to take an `Allocator` for the returned string and add a reference qualification. Add an rvalue-ref overload of `str()`. Change the `str(s)` overload to take an allocator template parameter that could differ from the buffer's own `Allocator`. Add a `str(s)` overload that moves from its string and a `view()` member function:

```
// ??, members:
basic_stringbuf<charT, traits, Allocator>* rdbuf() const;
```

```

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

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

```

6.4.1 30.8.4.1 basic_stringstream constructors [stringstream.cons]

Change the constructor specification to allow a string copy with a different allocator.

```

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

```

- 1 *Effects:* Constructs an object of class `basic_stringstream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which)`.

Add the following constructor specification:

```

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

```

- 2 *Effects:* Constructs an object of class `basic_stringstream<charT, traits>`, initializing the base class with `basic_stream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(str), which)` (30.8.2.1).

6.4.2 30.8.4.3 Member functions [stringstream.members]

Add the allocator parameter to the following `str()` overloads:

```

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

```

- 1 *Returns:* `rdbuf()->str(sa)`.

```

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

```

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

Add the following specifications:

```

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

```

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

```

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

```

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

5 [*Note*: Calling this member function leaves the stream object in a usable state with an emptied
underlying `basic_stringbuf`. — *end note*]

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

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

7 Appendix: Example Implementations

The given specification has been implemented within a recent version of the `sstream` header of gcc6. 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.