p0408r2 - Efficient Access to basic_stringbuf’s Buffer

Peter Sommerlad
2017-06-07

<table>
<thead>
<tr>
<th>Document Number:</th>
<th>p0408r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>2017-06-07</td>
</tr>
<tr>
<td>Project:</td>
<td>Programming Language C++</td>
</tr>
<tr>
<td>Audience:</td>
<td>LWG/LEWG</td>
</tr>
<tr>
<td>Target:</td>
<td>C++20</td>
</tr>
</tbody>
</table>

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].

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

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

— 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)

2.1.2 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, 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

— 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.

5.2 Open Issues discussed by LEWG in Issaquah

— 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.

— 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:

```cpp
// ??, 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, SAlloc>& 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. 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 `pilfer()` member function obtaining a string from the internal buffer by moving from it. Add the `view()` member function obtaining a `string_view` to the underlying internal buffer.

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

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

### 6.1.1 30.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Modify the following constructor specification:

```cpp
template<class SAlloc=Allocator>
```
explicit basic_stringbuf(
    const basic_string<charT, traits, Allocator>& str,
    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() (??), 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);

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.

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

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

    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);

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

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 epptr() >= 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 \( \text{gptr}() == \text{eback}() \) and \( \text{egptr}() == \text{eback}() + \text{size} \) hold.

\[ \text{basic_string}<\text{charT}, \text{traits}, \text{Allocator}> \text{pilfer();} \]

**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(\text{eback}(), \text{egptr}()-\text{eback}()). If the basic\_stringbuf was created with which \& \text{ios\_base::out} being true then basic\_string(\text{pbase}(), \text{high\_mark}-\text{pbase}()), where \text{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.

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

\[ \text{basic\_string\_view}<\text{charT}, \text{traits}> \text{view() const;} \]

**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(\text{eback}(), \text{egptr}()-\text{eback}()). If the basic\_stringbuf was created with which \& \text{ios\_base::out} being true then basic\_string\_view(\text{pbase}(), \text{high\_mark}-\text{pbase}()), where \text{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.

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

### 6.2 30.8.3 Adjust synopsis of basic\_istringstream [istringstream]

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

```cpp
// ??, constructors:
explicit basic\_istringstream(
   \text{ios\_base::openmode} \text{which} = \text{ios\_base::in});
\text{template<class SAlloc=\text{Allocator}>}
explicit basic\_istringstream(
   \text{const basic\_string<\text{charT}, \text{traits}, SAlloc>& str},
   \text{ios\_base::openmode} \text{which} = \text{ios\_base::in});
```
explicit basic_istringstream(
    basic_string<charT, traits, Allocator>&& str,
    ios_base::openmode which = ios_base::in);

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

Change the str() member function to allow different allocator argument for the new string to be used or the obtained string copy. Add an overload of the str(s) member function that moves from a string and add pilfer() and 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, SAlloc>& s);
    void str(basic_string<charT, traits, Allocator>&& s);
    basic_string_view<charT, traits> view() const;

6.2.1 30.8.3.1 basic_istringstream constructors [istringstream.cons]
Change the constructor specification to allow a string copy with a different allocator.

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

    Effects: Constructs an object of class basic_istringstream<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_istringstream(
        const basic_string<charT, traits, Allocator>&& str,
        ios_base::openmode which = ios_base::in);

    Effects: Constructs an object of class basic_istringstream<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 [iostreamstream.members]
Add the allocator parameter to the following str() overloads:

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

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

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

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

Add the following specifications:

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

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

basic_string<charT, traits, Allocator> pilfer();

Returns: rdbuf()->pilfer().

basic_string_view<charT, traits> view() const;

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 string:

// ??, constructors:
explicit basic_ostringstream(
    ios_base::openmode which = ios_base::out);

template<class SAlloc=Allocator>
explicit basic_ostringstream(
    const basic_string<charT, traits, Allocator>& 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 str() member function to allow different allocator argument for the new string to be used or the obtained string copy. Add an overload of the str(s) member function that moves from a string and add pilfer() and 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, Allocator>&& s);

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

basic_string<charT, traits, Allocator> pilfer();

basic_string_view<charT, traits> view() const;

6.3.1 30.8.4.1 basic_ostringstream constructors [sstream.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, SAlloc>& str,
  ios_base::openmode which = ios_base::out);

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);

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 [ostringstream.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;

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

// ??, constructors:
explicit basic_stringstream(
  ios_base::openmode which = ios_base::out | ios_base::in);
template<class SAlloc=Allocator>
explicit basic_stringstream(

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, Allocator>& 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 str() member function to allow different allocator argument for the new string to be
used or the obtained string copy. Add an overload of the str(s) member function that moves from
a string and add pilfer() and 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, SAlloc>& s);
void str(basic_string<charT, traits, Allocator>&& s);
basic_string<charT,traits,Allocator> pilfer();
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, SAlloc>& str,
    ios_base::openmode which = ios_base::out | ios_base::in);

Effects: Constructs an object of class basic_stringstream<charT, traits>, initializing
the base class with basic_iostream(&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);

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,SAlloc> str(const SAlloc& sa=SAlloc()) const;
Returns: `rdbuf()->str(sa)`.

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

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

Add the following specifications:

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

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

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

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

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

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_string-
buf_efficient/src.