p0408r0 - Efficient Access to basic_stringbuf's Buffer

Peter Sommerlad

2016-07-01

Document Number: p0408r0	
Date:	2016-07-01
Project:	Programming Language C++
Audience:	LWG/LEWG

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 pessimissation alive on `basic_stringbuf`.

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 &&)`.

3 Acknowledgements

- Daniel Krügler encouraged me to pursue this track.

4 Impact on the Standard

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

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 to be Discussed by LEWG / LWG

- Is the name of the `str_view()` member function ok?
- Should the `str()&&` overload be provided for move-out?
- Should `str()&&` empty the character sequence or leave it in an unspecified but valid state?

6 Technical Specifications

The following is relative to n4594.

6.1 27.8.2 Adjust synopsis of basic_stringbuf [stringbuf]

Add a new constructor overload:

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

Change the `const`-overload of `str()` member function to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

Add two overloads of the `str()` member function and add the `str_view()` member function:

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

6.1.1 27.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Add the following constructor specification:

```
explicit basic_stringbuf(
  basic_string<charT, traits, Allocator>&& s,
  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()` (27.6.3.1), and initializing mode with which. Then calls `str(std::move(s))`.

6.1.2 27.8.2.3 Member functions [stringbuf.members]

Change the `const`-overload of `str()` member function specification to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

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

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

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

2 *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 `gptr() == eback()` and `egptr() == eback() + size` hold.

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

3 *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 an empty `basic_string` is returned.

4 *Postcondition:* The underlying character sequence is empty.

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

5 *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 a `basic_string_view` referring to an empty range is returned.

6 [*Note:* Using the returned object after destruction or any modification of `*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 27.8.3 Adjust synopsis of basic_istringstream [istringstream]

Add a new constructor overload:

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

Change the `const`-overload of `str()` member function to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

Add an overload of the `str()` member function and add the `str_view()` member function:

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

6.2.1 27.8.3.1 basic_istringstream constructors [istringstream.cons]

Add the following constructor specification:

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

1 *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))` (27.8.2.1).

6.2.2 27.8.3.3 Member functions [istringstream.members]

Change the `const`-overload of `str()` member function specification to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

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

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

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

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

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

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

3 *Returns:* `rdbuf()->str_view()`.

6.3 27.8.4 Adjust synopsis of basic_ostringstream [ostringstream]

Add a new constructor overload:

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

Change the `const`-overload of `str()` member function to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

Add an overload of the `str()` member function and add the `str_view()` member function:

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

6.3.1 27.8.4.1 basic_ostringstream constructors [ostringstream.cons]

Add the following constructor specification:

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

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>(std::move(str), which | ios_base::in))` (27.8.2.1).

6.3.2 27.8.4.3 Member functions [ostringstream.members]

Change the `const`-overload of `str()` member function specification to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

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

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

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

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

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

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

3 *Returns:* `rdbuf()->str_view()`.

6.4 27.8.5 Adjust synopsis of basic_stringstream [stringstream]

Add a new constructor overload:

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

Change the `const`-overload of `str()` member function to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

Add an overload of the `str()` member function and add the `str_view()` member function:

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

6.4.1 27.8.4.1 basic_stringstream constructors [stringstream.cons]

Add the following constructor specification:

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

1 *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 | ios_base::in))` (27.8.2.1).

6.4.2 27.8.4.3 Member functions [stringstream.members]

Change the `const`-overload of `str()` member function specification to add a reference qualification. This avoids ambiguities with the rvalue-ref overload of `str()`.

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

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

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

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

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

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

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

3 *Returns:* `rdbuf()->str_view()`.

7 Appendix: Example Implementations

The given specification has been implemented within a recent version of the sstream header of gcc6. Here are some definitions taken from there:

// basic_stringbuf:

```
      explicit
      basic_stringbuf(__string_type&& __str,
                      ios_base::openmode __mode = ios_base::in | ios_base::out)
      : __streambuf_type(), _M_mode(), _M_string(std::move(__str))
      { _M_stringbuf_init(__mode); }
        using __string_view_type=experimental::basic_string_view<_CharT,_Traits>;

      __string_view_type str_view() const {
        __string_view_type __ret{};
        if ( this->pptr())  {
            // The current egptr() may not be the actual string end.
            if (this->pptr() > this->egptr())
              __ret = __string_view_type(this->pbase(), this->pptr()-this->pbase());
            else
              __ret = __string_view_type(this->pbase(), this->egptr()-this->pbase());
          }
        else {
                __ret = _M_string;
        }
        return __ret;
      }
     void
      str(__string_type&& __s)
      {
        _M_string.assign(std::move(__s));
        _M_stringbuf_init(_M_mode);
      }
```

//basic_istringstream

```
      explicit
      basic_istringstream(__string_type&& __str,
                          ios_base::openmode __mode = ios_base::in)
      : __istream_type(), _M_stringbuf(std::move(__str), __mode | ios_base::in)
      { this->init(&_M_stringbuf); }
        using __string_view_type=experimental::basic_string_view<_CharT,_Traits>;
```

```
      __string_view_type
      str_view() const
      { return _M_stringbuf.str_view(); }
      void
      str( __string_type&& __s)
      { _M_stringbuf.str(std::move(__s)); }

  //basic_ostringstream
      explicit
      basic_ostringstream(__string_type&& __str,
                          ios_base::openmode __mode = ios_base::out)
      : __ostream_type(), _M_stringbuf(std::move(__str), __mode | ios_base::out)
      { this->init(&_M_stringbuf); }
        using __string_view_type=experimental::basic_string_view<_CharT,_Traits>;
      __string_view_type
      str_view() const
      { return _M_stringbuf.str_view(); }
      void
      str( __string_type&& __s)
      { _M_stringbuf.str(std::move(__s)); }

  //basic_stringstream
      explicit
      basic_stringstream( __string_type&& __str,
                         ios_base::openmode __m = ios_base::out | ios_base::in)
      : __iostream_type(), _M_stringbuf(std::move(__str), __m)
      { this->init(&_M_stringbuf); }
        using __string_view_type=experimental::basic_string_view<_CharT,_Traits>;
      __string_view_type
      str_view() const
      { return _M_stringbuf.str_view(); }
      void
      str( __string_type&& __s)
      { _M_stringbuf.str(std::move(__s)); }
```