p0407r2 - Allocator-aware basic_stringbuf

Peter Sommerlad

2017-11-16

1 History

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 allocator support is only available through a template parameter but not really encouraged or allowed on a per-object basis. The second issue 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. There is a second paper p0408 on the efficient internal buffer access that sidesteps the extra copy.

1.1 n3172

In Batavia 2010 n3172 was discussed and the issue LWG 2429 was closed with NAD but including an encouraging note that n3172 was just not enough (retrospectively this was due to the rush to get C++11 done). And there was not yet the allocator infrastructure in place that we aim for with C++17.

1.2 p0407r0

Since this paper has not yet really been discussed by LEWG or LWG this update mainly reflects the re-ordering of the standard’s section numbers. It also adds access to the set allocator of basic_stringbuf.

1.3 p0407r1

Thanks to sweat-dripping effort by Pablo Halpern, I was able to come up with an easier way to correctly specify allocator support by introducing an addition exposition-only basic_string member variable to the basic_stringbuf. Also added an overload of the move constructor with an additional allocator parameter.
The paper p0408 (efficient buffer access) needs to be slightly adopted according to this paper if accepted before this paper (this paper passed p0408 in LEWG in Albuquerque).

2 Introduction

This paper proposes one adjustment to basic_stringbuf and the corresponding stream class templates to enable the actual use of allocators. It follows the direction of what basic_string provides and thus allows implementations who actually use basic_string as the internal buffer for basic_stringbuf to directly map the allocator to the underlying basic_string.

3 Acknowledgements

— Thanks go to Pablo Halpern who originally started this, helped with getting Allocator support simpler to specify correctly, and Daniel Krügler who pointed this out to me and told me to split the two issues into two independent papers. Also to Alisdair Meredith for feedback on earlier versions pointing out deficiencies with respect to my abilities to specify allocators.

4 Motivation

With the introduction of more useful allocator API in the recent editions of the standard including the planned C++17, it is more desirable to have the library classes that allocate and release memory to employ that infrastructure, e.g., to provide thread-specific allocation that can work without employing mutual exclusion. Unfortunately streams based on strings do not take allocator object arguments, whereas they already have the corresponding template parameter. This seems to be an easy to provide extension that almost looks overlooked by previous allocator-specific adaptations of the standard’s text.

5 Impact on the Standard

This is an extension to the constructor API of basic_stringbuf, basic_stringstream, basic_istringstream, and basic_ostringstream class templates to follow the constructors taking allocators from basic_string. Because each constructor is extended with a parameter as the last one and this parameter is provided with a default argument there should be minimal impact on existing client code. Regular usage should be completely unaffected.

The class basic_stringbuf gets an additional member function to obtain a copy of the underlying
allocator object, like `basic_string` provides.

6 Design Decisions

6.1 General Principles

Allocator support in the standard library is lacking for string-based streams and seems to be addable in a straightforward way, because all class templates already take it as template parameter.

6.2 Open Issues to be Discussed by LEWG / LWG

— Does it make sense to add `noexcept` specifications for `move()` and `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.

— Are there other functions with respect to string streams that would require an allocator parameter? I do not think so, except move (from string) constructors in P0408 which require additional overloads along the lines of the constructors extended here.

— There is some overlap with P0408, how to deal with that overlap?

7 Technical Specifications

7.1 30.8.2 Adjust synopsis of `basic_stringbuf` [stringbuf]

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. 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`?

```cpp
explicit basic_stringbuf(
    ios_base::openmode which = ios_base::in | ios_base::out,
    const Allocator &a = Allocator());

template<class SAllocator>
explicit basic_stringbuf(
    const basic_string<CharT, Traits, SAllocator>& str,
    ios_base::openmode which = ios_base::in | ios_base::out,
    const Allocator& a = Allocator());

template<class SAllocator>
explicit basic_stringbuf(
    const basic_string<CharT, Traits, SAllocator>& str,
    const Allocator& a) :
    basic_stringbuf(std,ios_base::openmode(ios_base::in | ios_base::out),a) {}

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

basic_stringbuf(basic_stringbuf&& rhs, const Allocator& a);
```
Add the following declaration to the public section of synopsis of the class template `basic_stringbuf`:

```cpp
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.

```cpp
private:
    ios_base::openmode mode; // exposition only
    basic_string<charT, traits, Allocator> buf; // exposition only
```

May be: Add a conditional noexcept specification to `swap` based on Allocator’s behaviour?:

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

Append a list item p2.2 and a paragraph p3 to the text following the synopsis:

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 `char` objects. The sequence can be initialized from, or made available as, an object of class `basic_string`.

2. In every specialization `basic_stringbuf<charT, traits, Allocator>`, the type `allocator_traits<Allocator>::value_type` shall name the same type as `charT`. Every object of type `basic_stringbuf<charT, traits, Allocator>` shall use an object of type `Allocator` to allocate and free storage for the internal buffer of `charT` objects as needed. The `Allocator` object used shall be obtained as described in 23.2.1 [container.requirements.general].

3. For the sake of exposition, the maintained data is presented here as:

   - `ios_base::openmode mode`, has in set if the input sequence can be read, and out set if the output sequence can be written.
   - `basic_string<charT, traits, Allocator> buf` holds the underlying character sequence.

### 7.1.1 30.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Adjust the constructor specifications taking the additional `Allocator` parameter and an overload for the move-constructor taking an `Allocator`:

```cpp
explicit basic_stringbuf(
    ios_base::openmode which = ios_base::in | ios_base::out,
    const Allocator &a = Allocator());
```

1. 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 `a`.

2. Postconditions: `str() == ""`. 
template<class SAllocator>
explicit basic_stringbuf(
    const basic_string<charT, traits, SAllocator>& s,
    ios_base::openmode which = ios_base::in | ios_base::out,
    const Allocator& a = Allocator());

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

basic_stringbuf(basic_stringbuf&& rhs);

Effects: Move constructs from the rvalue rhs. In the first form buf is initialized from
\{std::move(rhs.buf)\}. In the second form buf is initialized from \{std::move(rhs.buf), a\}. It is implementation-defined whether the sequence pointers in *this (eback(), gptr(),
egptr(), pbase(), pptr(), eptr()) 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.

Postconditions: 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.

- \(\text{str}() == \text{rhs}_p.\text{str}()\)
- \(\text{gptr()} - \text{eback()} == \text{rhs}_p.\text{gptr()} - \text{rhs}_p.\text{eback()}\)
- \(\text{egptr()} - \text{eback()} == \text{rhs}_p.\text{egptr()} - \text{rhs}_p.\text{eback()}\)
- \(\text{pptr()} - \text{pbase()} == \text{rhs}_p.\text{pptr()} - \text{rhs}_p.\text{pbase()}\)
- \(\text{epptr()} - \text{pbase()} == \text{rhs}_p.\text{epptr()} - \text{rhs}_p.\text{pbase()}\)
- if (eback()) eback() != rhs_a.eback()
- if (gptr()) gptr() != rhs_a.gptr()
- if (egptr()) egptr() != rhs_a.egptr()
- if (pbase()) pbase() != rhs_a.pbase()
- if (pptr()) pptr() != rhs_a.pptr()
- if (epptr()) epptr() != rhs_a.epptr()

7.2 30.8.2.3 Assign and swap [stringbuf.assign]

basic_stringbuf& operator=(basic_stringbuf&& rhs);

Effects: Move assigns buf from std::move(rhs.buf). After the move assignment *this has
the observable state it would have had if it had been move constructed from rhs (see ??).

Returns: *this.

void swap(basic_stringbuf& rhs);

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).

7.3 30.8.2.3 Member functions [stringbuf.members]

Add the definition of the get_allocator function:

allocator_type get_allocator() const noexcept;

Returns: buf.get_allocator().

7.4 30.8.3 Adjust synopsis of basic_istringstream [istringstream]

Change each of the non-move, non-deleted constructors to add a const-ref Allocator parameter as last parameter with a default constructed Allocator as default argument. Allow a string with a different allocator type here as well.

explicit basic_istringstream(
    ios_base::openmode which = ios_base::in,
    const Allocator& a = Allocator());

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

template <class SAllocator>
explicit basic_istringstream(
    const basic_string<charT, traits, SAllocator>& str,
    const Allocator& a)
    : basic_istringstream(str, ios_base::in, a) {}

7.4.1 30.8.3.1 basic_istringstream constructors [istringstream.cons]

Adjust the constructor specifications taking the additional Allocator parameter and adjust the delegation to basic_stringbuf constructors in the Effects clauses in p1 and p2 to pass on the given allocator object.

explicit basic_istringstream(
    ios_base::openmode which = ios_base::in,
    const Allocator& a = Allocator());

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>(which | ios_base::in, a) (30.8.2.1).

template <class SAllocator>
explicit basic_istringstream(
    const basic_string<charT, traits, SAllocator>& str,
    ios_base::openmode which = ios_base::in,
    const Allocator& a = Allocator());
**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, a)` (30.8.2.1).

### 7.5 30.8.4 Adjust synopsis of `basic_ostringstream` [sstream]

Change each of the non-move, non-deleted constructors to add a const-ref `Allocator` parameter as last parameter with a default constructed `Allocator` as default argument.

```cpp
explicit basic_ostringstream(
    ios_base::openmode which = ios_base::out,
    const Allocator& a = Allocator());
```

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

```cpp
template <class SAllocator>
explicit basic_ostringstream(
    const basic_string<charT, traits,SAllocator>& str,
    const Allocator& a)
    : basic_ostringstream(std, ios_base::out, a) {}  
```

### 7.5.1 30.8.4.1 `basic_ostringstream` constructors [sstream.cons]

Adjust the constructor specifications taking the additional `Allocator` parameter and adjust the delegation to `basic_stringbuf` constructors in the Effects clauses in p1 and p2 to pass on the given allocator object.

```cpp
explicit basic_ostringstream(
    ios_base::openmode which = ios_base::out,
    const Allocator& a = Allocator());
```

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

```cpp
template <class SAllocator>
explicit basic_ostringstream(
    const basic_string<charT, traits,SAllocator>& str,
    const Allocator& a)
    : basic_ostringstream(std, ios_base::out, a) {}  
```

### 7.5.2 30.8.4.1 `basic_ostringstream` constructors [sstream.cons]

```cpp
explicit basic_ostringstream(
    const basic_string<charT, traits,SAllocator>& str,
    const Allocator& a)
    : basic_ostringstream(std, ios_base::out, a) {}  
```

**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, a)` (30.8.2.1).

### 7.6 30.8.5 Adjust synopsis of `basic_stringstream` [sstream]

Change each of the non-move, non-deleted constructors to add a const-ref `Allocator` parameter as last parameter with a default constructed `Allocator` as default argument.

```cpp
explicit basic_stringstream(
```
```cpp
ios_base::openmode which = ios_base::out | ios_base::in,
const Allocator& a = Allocator());

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

template <class SAllocator>
explicit basic_stringstream(
const basic_string<
charT, traits, SAllocator>& str,
const Allocator& a)
: basic_stringstream(std, ios_base::openmode(ios_base::in | ios_base::out), a) {}
```

### 7.6.1 30.8.5.1 basic_stringstream constructors [stringstream.cons]

Adjust the constructor specifications taking the additional Allocator parameter and adjust the delegation to basic_stringbuf constructors in the Effects clauses in p1 and p2 to pass on the given allocator object.

```cpp
explicit basic_stringstream(
ios_base::openmode which = ios_base::out | ios_base::in,
const Allocator& a = Allocator());
```

**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>(which, a)`.

```cpp
template <class SAllocator>
explicit basic_stringstream(
const basic_string<
charT, traits, SAllocator>& str,
const Allocator& a)
: basic_stringstream(std, ios_base::openmode(ios_base::in | ios_base::out), a) {}
```

**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, a)`.

### 8 Appendix: Example Implementations

An implementation of the additional constructor parameter was done by the author in the `<sstream>` header of gcc 6.1. It seems trivial, since all significant relevant usage is within `basic_string`.