Doc No: N3172=10-0162
Date: 2010-10-18
Authors: Pablo Halpern
Intel Corp..
phalpern@halpernwightsoftware.com

Allocators for stringstream (US140)

Contents

National Body Comments and Issues

This paper proposes a complete resolution for comment US 140 to the July, 2010 FCD.

Document Conventions

All section names and numbers are relative to the August 2010 WP, **N3126**.

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with ~~red strikeouts for deleted text~~ and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

Requests for LWG opinions and guidance appear with light (yellow) shading. It is expected that changes resulting from such guidance will be minor and will not delay acceptance of this proposal in the same meeting at which it is presented.

Background

A great deal of effort has been put into making allocators in C++0x useful and powerful. The uniform use of allocators can make a program more efficient (in space and/or time) or more flexible. To be maximally useful, allocators must be consistently and uniformly available for use in the interface of any general-purpose library, in much the way that `const`, in order to be maximally useful, had to be used consistently in library interfaces (thanks to Matt Austern for the preceding observation). All of the standard containers, `basic_string`, `shared_ptr`, and `function` use allocators to allocate memory and thus give programmers control over how

their own programs use memory. However, `stringstream` and parts of `regex` stand out as two classes that do not have interfaces that allow the user to supply his/her own allocator. Having just these two exceptions is a defect, as per national body comments US 140 (`stringstream`) and US 104 (`regex`). The lack of allocator use by `stringstream` is especially troubling because `stringstream` is already template on an `Allocator` parameter and because it manipulates strings, which use allocators. (The `regex` component, US 104, is being addressed in a separate paper, N3171, by Mike Spertus.)

This paper proposes wording changes to the definition of the `basic_stringstream` class template that will make it consistent with the intended use of allocators. The template already has an allocator parameter (used for instantiating `basic_string`) that can readily be used to specify an allocator type for the `stringstream` itself. Thus, what is being proposed here is a minimally-invasive fix having no effect on programmers that don't use allocators. The `stringstream` object and the `basic_string` objects that it manipulates must use the same allocator type, according to this proposal. It is my belief that this covariance is desirable.

Proposed Wording

In section 27.8.1 [stringbuf] add a description of allocator usage as follows:

> **27.8.1 Class template `basic_streambuf` [stringbuf]**
>
> Instantiations of the class template `basic_streambuf` manage a sequence of char-like objects. The sequence is allocated and deallocated using an allocator (`alloc`) in much the same way that `basic_string` does (21.4) [basic.string]. Specifically, the sequence is allocated using `allocator_traits<Allocator>::allocate` and deallocated using `allocator_traits<Allocator>::deallocate`. The char-like objects are not constructed or destroyed using `allocator_traits<Allocator>::construct` and `allocator_traits<Allocator>::destroy`.

Add or modify the following constructors:

```
// 27.8.1.1 Constructors:
explicit basic_stringbuf(ios_base::openmode which
                           = ios_base::in | ios_base::out,
                         const Allocator& a = Allocator());
explicit basic_stringbuf
(const basic_string<charT,traits,Allocator>& str,
 ios_base::openmode which = ios_base::in | ios_base::out);
basic_stringbuf(const basic_string<charT,traits,Allocator>& str,
                ios_base::openmode which, const Allocator& a);
basic_stringbuf(basic_stringbuf&& rhs);
basic_stringbuf(basic_stringbuf&& rhs, const Allocator& a);
```

Add a new accessors:

```
// 27.8.1.3 Get and set
```

```
    Allocator get_allocator() const;
    basic_string<charT,traits,Allocator>&
        copy_str(basic_string<charT,traits,Allocator>& s) const;
```

And add a new data member for exposition:

```
    ios_base::openmode mode; // exposition only
    Allocator          alloc; // exposition only
```

In section 27.8.1.1 [stringbuf.cons], add or modify constructor descriptions:

27.8.1.1 basic_stringbuf constructors [stringbuf.cons]

```
explicit basic_stringbuf(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()` (27.6.2.1), ~~and~~ initializing `mode` with `which`, and initializing `alloc` with `a`.

Postcondition: `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()` (27.6.2.1), ~~and~~ initializing `mode` with `which`, and initializing `alloc` with `allocator_traits<Allocator>::select_on_container_copy_construction (s.get_allocator())`. Then ~~calls str(s)~~ initializes the input and output sequences as if by calling `str(basic_string<charT,traits,Allocator>(s,alloc))`.

The allocator propagation works as if we were copy constructing from `s`, even though `*this` and `s` are of different types. The definition of setting the input and output sequence is too complicated to repeat here, hence the call to `str()`. We need to change this call from the WP version because, having set the allocator appropriately, it is necessary to construct a string with the same allocator or else the call to `str()` might change it. A real implementation would avoid the extra string construction and, instead, call a private function that is also called by `str()`.

```
basic_stringbuf(const basic_string<charT,traits,Allocator>& str,
                ios_base::openmode which, const Allocator& a);
```

Effects: Constructs an object of class `basic_stringbuf`, initializing the base class with `basic_streambuf()` (27.6.2.1), initializing `mode` with `which`, and initializing `alloc` with a. Then initializes the input and output sequences as if by calling `str(basic_string<charT,traits,Allocator>(s,alloc))`.

```
basic_stringbuf(basic_stringbuf&& rhs);
```

Effects: Move constructs from the rvalue `rhs`. 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.~~

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.

— `str() == rhs_p.str()`
— `mode == rhs_p.mode`
— `get_allocator() == rhs_p.get_allocator()`
— `gptr() - eback() == rhs_p.gptr() - rhs_p.eback()`
— `egptr() - eback() == rhs_p.egptr() - rhs_p.eback()`
— `pptr() - pbase() == rhs_p.pptr() - rhs_p.pbase()`
— `epptr() - pbase() == rhs_p.epptr() - rhs_p.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()`

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

Effects: Move constructs from the rvalue `rhs`, initializing `alloc` to `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. [*Note:* the sequence pointers in this will never obtain the values which `rhs` had if `a != rhs.get_allocator()`, but might if the allocators compared equal. – *end note*]

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.

— `str() == rhs_p.str()`
— `mode == rhs_p.mode`
— `get_allocator() == a`
— `gptr() - eback() == rhs_p.gptr() - rhs_p.eback()`
— `egptr() - eback() == rhs_p.egptr() - rhs_p.eback()`
— `pptr() - pbase() == rhs_p.pptr() - rhs_p.pbase()`
— `epptr() - pbase() == rhs_p.epptr() - rhs_p.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()`

In section 27.8.1.2 [stringbuf.assign], change the descriptions of assign and member swap:

27.8.1.2 Assign and swap [stringbuf.assign]

`basic_stringbuf& operator=(basic_stringbuf&& rhs);`

Effects: If `allocator_traits<Allocator>::propagate_on_move_assignment::value` is true,

then, ~~A~~after the move assignment, `*this` has the observable state it would have had if it had been move constructed from `rhs`; otherwise it has the same observable state it would have had if it had been constructed from `rhs` and `alloc` (see 27.8.1.1) (i.e., the allocator is not assigned unless `Allocator` is specifically designated to propagate during move assignment).

Returns: *this.

```
void swap(basic_stringbuf& rhs);
```

Preconditions: either `allocator_traits<Allocator>::propagate_on_swap::value` is true, or `this->alloc == rhs.alloc`.

Effects: Exchanges the state of `*this` and `rhs`.

In 27.8.1.3, add the definition of the new accessors and change the descriptions of `str()`:

```
Allocator get_allocator() const;
```

Returns: `alloc`

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

~~*Returns*: A basic_string object whose content is equal to the~~
Effects: Sets the contents of `s` 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 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 `s` is set to a zero length `basic_string` ~~is returned~~.

Returns: `s`

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

Effects: Equivalent to:

```
Allocator a = allocator_traits<Allocator>::
    select_on_container_copy_construction(alloc);
basic_string<charT,traits,Allocator> s(a);
return copy_str(s);
```

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

Effects: Copies the content of `s` into the `basic_stringbuf` underlying character sequence and initializes the input and output sequences according to mode. If `allocator_traits<Alloc>::propagate_on_container_copy_assignment::value` is `true`, then also assign the value of `s.get_allocator()` to `alloc`.

Postconditions: If `mode & ios_base::out` is `true`, `pbase()` points to the first underlying character and `epptr() >= pbase() + s.size()` holds; in addition, if `mode & ios_base::in` is `true`, `pptr() == pbase() + s.data()` 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() + s.size()` hold.

Basically, the `str(s)` function is being treated as a heterogeneous assignment from `s` to `*this`, as far as the allocator is concerned.

In section 27.8.2, add or modify the following constructors and add two new accessors:

```
// 27.8.2.1 Constructors:
explicit basic_istringstream(ios_base::openmode which = ios_base::in,
                             const Allocator& a = Allocator());
explicit basic_istringstream(
       const basic_string<charT,traits,Allocator>& str,
       ios_base::openmode which = ios_base::in);
basic_istringstream(
       const basic_string<charT,traits,Allocator>& str,
       ios_base::openmode which, const Allocator& a);
basic_istringstream(basic_istringstream&& rhs);
basic_istringstream(basic_istringstream&& rhs, const Allocator& a);

// 27.8.2.3 Members
Allocator get_allocator() const;
basic_string<charT,traits,Allocator>&
    copy_str(basic_string<charT,traits,Allocator>& s) const;
```

In section 27.8.2.1 [istringstream.cons], add the new constructor descriptions:

27.8.2.1 basic_istringstream constructors [istringstream.cons]

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

Effects: Constructs an object of class `basic_istringstream<charT, traits,Allocator>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(which | ios_base::in, a)` (27.8.1.1).

Note: the `Allocator` template parameter is not new; it was just inadvertently omitted in the previous draft. The `a` *function* parameter is new.

```
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,Allocator>, initializing the base class with basic_istream(&sb) and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::in)` (27.8.1.1).

```
basic_istringstream(const basic_string<charT,traits,Allocator>& str,
                    ios_base::openmode, const Allocator& a);
```

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)` (27.8.1.1).

```
basic_istringstream(basic_istringstream&& rhs);
```

Effects: Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_istream<charT,traits,Allocator>::set_rdbuf(&sb)` is called to install the contained basic_stringbuf.

```
basic_istringstream(basic_istringstream&& rhs, const Allocator& a);
```

Effects: Constructs an object of class `basic_istringstream<charT, traits,Allocator>`, initializing the base class with `basic_istream(std::move(rhs))`, initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(rhs.sb), a)`, and installing `sb` by calling `set_rdbuf(&sb)`.

And define the accessors:

```
Allocator get_allocator() const;
```

Returns: `sb.get_allocator()`

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

Returns: `sb.copy_str(s)`

In section 27.8.3 [ostringstream], add and modify constructors and add two new accessors:

```
// 27.8.3.1 Constructors/destructor:
explicit basic_ostringstream(ios_base::openmode which = ios_base::out,
                             const Allocator& a = Allocator());
explicit basic_ostringstream(
         const basic_string<charT,traits,Allocator>& str,
         ios_base::openmode which = ios_base::out);
basic_ostringstream(const basic_string<charT,traits,Allocator>& str,
                    ios_base::openmode which, const Allocator& a);
basic_ostringstream(basic_ostringstream&& rhs);
basic_ostringstream(basic_ostringstream&& rhs, const Allocator& a);

// 27.8.3.3 Members
Allocator get_allocator() const;
basic_string<charT,traits,Allocator>&
    copy_str(basic_string<charT,traits,Allocator>& s) const;
```

In section 27.8.3.1 [ostringstream.cons], define the constructors:

```
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)` (27.8.1.1).

```
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(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::out)` (27.8.1.1).

```
basic_ostringstream(const basic_string<charT,traits,Allocator>& str,
                    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(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::out, a)` (27.8.1.1).

```
basic_ostringstream(basic_ostringstream&& rhs);
```

Effects: Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_ostream<charT,traits,Allocator>::set_rdbuf(&sb)` is called to install the contained `basic_stringbuf`.

```
basic_ostringstream(basic_ostringstream&& rhs, const Allocator& a);
```

Effects: Constructs an object of class `basic_ostringstream<charT, traits,Allocator>`, initializing the base class with `basic_ostream(std::move(rhs))`, initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(rhs.sb), a)`, and installing `sb` by calling `set_rdbuf(&sb)`.

And define the accessors:

```
Allocator get_allocator() const;
```

Returns: `sb.get_allocator()`

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

Returns: `sb.copy_str(s)`

In section 27.8.4 [stringstream], add/ modify constructors and add two new accessors:

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

```
basic_stringstream(const basic_string<charT,traits,Allocator>& str,
                   ios_base::openmode which, const Allocator& a);
basic_stringstream(basic_stringstream&& rhs);
basic_stringstream(basic_stringstream&& rhs, const Allocator& a);

// 27.8.3.3 Members
Allocator get_allocator() const;
basic_string<charT,traits,Allocator>&
    copy_str(basic_string<charT,traits,Allocator>& s) const;
```

In section 27.8.5 [stringstream.cons] (Note to the editor: This section should be renumbered 27.8.4.1 to be consistent with the other stream types.) , define the new constructors:

27.8.5 basic_stringstream constructors [stringstream.cons]

```
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, Allocator>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_stringbuf<charT,traits,Allocator>(which, a)`.

```
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(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which)`.

```
basic_stringstream(const basic_string<charT,traits,Allocator>& str,
                   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(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which, a)`.

```
basic_stringstream(basic_stringstream&& rhs);
```

Effects: Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_istream<charT,traits,Allocator>::set_rdbuf(&sb)` is called to install the contained `basic_stringbuf`.

```
basic_stringstream(basic_stringstream&& rhs, const Allocator& a);
```

Effects: Constructs an object of class `basic_stringstream<charT, traits,Allocator>`, initializing the base class with `basic_iostream(std::move(rhs))`, initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(std::move(rhs.sb), a)`, and installing `sb` by calling `set_rdbuf(&sb)`.

And define the accessors:

```
Allocator get_allocator() const;
```

Returns: `sb.get_allocator()`

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

Returns: `sb.copy_str(s)`

Acknowledgements

Thanks to John Lakos, who convinced me of the importance of writing this paper despite my desire to avoid doing the work.

References

N3102: ISO/IEC FCD 14882, C++0X, National Body Comments