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 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. There will be a second paper on the efficient internal buffer access.

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.

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 and Daniel Krügler who pointed this out to me and told me to split the two issues into two independent papers.

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.

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

- Do we need to say something about the effect of assignment and swap on the allocator?
- Are there other functions with respect to string streams that would require an allocator parameter? I do not think so.
7 Technical Specifications

7.1 27.8.2 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.

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

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

Append a paragraph p3 to the text following the synopsis:

1 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]. [Note: Implementations using basic_string internally, will simply pass the allocator parameter to the corresponding basic_string constructors. — end note]

7.1.1 27.8.2.1 basic_stringbuf constructors [stringbuf.cons]

Adjust the constructor specifications taking the additional Allocator parameter, no further explanation required:

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

and

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

7.2 27.8.3 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.
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,
    const Allocator &a=Allocator());

Append a paragraph p2 to the text following the synopsis:
1 In every specialization basic_istringstream<charT, traits, Allocator>, the type
  allocator_traits<Allocator>::value_type shall name the same type as charT. Ev-
  ery object of type basic_istringstream<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]. [Note: Implementations using basic_string inter-
  nally, will simply pass the allocator parameter to the corresponding basic_string con-
  structors. — end note]

7.2.1 27.8.3.1 basic_istringstream constructors [istreamstream.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());
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>(which | ios_base::in, a))(27.8.2.1).

explicit basic_istringstream(
    const basic_string<charT, traits, Allocator>& str,
    ios_base::openmode which = ios_base::in,
    const Allocator &a=Allocator());
2  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))(27.8.2.1).

7.3 27.8.4 Adjust synopsis of basic_ostringstream
      [ostringstream]
Change each of the non-move, non-deleted constructors to add a const-ref Allocator pa-
rameter as last parameter with a default constructed Allocator as default argument.
Append a paragraph p2 to the text following the synopsis:

In every specialization basic_ostringstream<charT, traits, Allocator>, the type allocator_traits<Allocator>::value_type shall name the same type as charT. Every object of type basic_ostringstream<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]. [Note: Implementations using basic_string internally, will simply pass the allocator parameter to the corresponding basic_string constructors. — end note]

7.3.1 27.8.4.1 basic_ostringstream constructors [ostringstream.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_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) (27.8.2.1).

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

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) (27.8.2.1).

7.4 27.8.5 Adjust synopsis of basic_stringstream [stringstream]

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

Append a paragraph p2 to the text following the synopsis:

In every specialization basic_stringstream<charT, traits, Allocator>, the type allocator_traits<Allocator>::value_type shall name the same type as charT. Every object of type basic_stringstream<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]. [Note: Implementations using basic_string internally, will simply pass the allocator parameter to the corresponding basic_string constructors. — end note]

7.4.1 27.8.5.1 basic_stringstream 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.

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

1 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).

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

2 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.