pXXXXr0 - A strstream replacement using a span<T>as buffer

Peter Sommerlad

2016-09-30

Document Number: pXXXXr0	(N2065 done right?)
Date:	2016-09-30
Project:	Programming Language C++
Audience:	LWG/LEWG

1 History

Streams have been the oldest part of the C++ standard library and especially strstreams that can use pre-allocated buffers have been deprecated for a long time now, waiting for a replacement. p0407 and p0408 provide the efficient access to the underlying buffer for stringstreams that strstream provided solving half of the problem that strstreams provide a solution for. The other half is using a fixed size pre-allocated buffer, e.g., allocated on the stack, that is used as the stream buffers internal storage.

A combination of external-fixed and internal-growing buffer allocation that strstreambuf provides is IMHO a doomed approach and very hard to use right.

There had been a proposal for the pre-allocated external memory buffer streams in N2065 but that went nowhere. Today, with `span<T>` we actually have a library type representing such buffers views we can use for specifying (and implementing) such streams. They can be used in areas where dynamic (re-)allocation of stringstreams is not acceptable but the burden of caring for a pre-existing buffer during the lifetime of the stream is manageable.

2 Introduction

This paper proposes a class template `basic_spanbuf` and the corresponding stream class templates to enable the use of streams on externally provided memory buffers. No ownership or re-allocation support is given. For those features we have string-based streams.

3 Acknowledgements

- Thanks go to Jonathan Wakely who pointed the problem of `strstream` out to me and to Neil Macintosh to provide the span library type specification.

4 Motivation

To finally get rid of the deprecated `strstream` in the C++ standard we need a replacement. p0407/p0408 provide one for one half of the needs for `strstream`. This paper provides one for the second half: fixed sized buffers.

[*Example:* reading input from a fixed pre-arranged character buffer:

```
char input[] = "10 20 30";
ispanstream is{span<char>{input}};
int i;
is >> i;
ASSERT_EQUAL(10,i);
is >> i ;
ASSERT_EQUAL(20,i);
is >> i;
ASSERT_EQUAL(30,i);
is >>i;
ASSERT(!is);
```

— *end example*] [*Example:* writing to a fixed pre-arranged character buffer:

```
char  output[30]{}; // zero-initialize array
ospanstream os{span<char >{output}};
os << 10 << 20 << 30 ;
auto const sp = os.span();
ASSERT_EQUAL(6,sp.size());
ASSERT_EQUAL("102030",std::string(sp.data(),sp.size()));
ASSERT_EQUAL(static_cast<void*>(output),sp.data()); // no copying of underlying data!
ASSERT_EQUAL("102030",output); // initialization guaranteed NUL termination
```

— *end example*]

5 Impact on the Standard

This is an extension to the standard library to enable deletion of the deprecated `strstream` classes by providing `basic_spanbuf`, `basic_spanstream`, `basic_ispanstream`, and `basic_ospanstream` class templates that take an object of type `span<charT>` which provides an external buffer to be used by the stream.

6 Design Decisions

6.1 General Principles

6.2 Open Issues to be Discussed by LEWG / LWG

- Should arbitrary types as template arguments to `span` be allowed to provide the underlying buffer by using the `byte` sequence representation `span` provides. (I do not think so, but someone might have a usecase.)

7 Technical Specifications

Insert a new section 27.x in chapter 27 after section 27.8 [string.streams]

7.1 27.x Span-based Streams [span.streams]

This section introduces a stream interface for user-provided fixed-size buffers.

7.1.1 27.x.1 Overview [span.streams.overview]

The header `<spanstream>` defines four class templates and eight types that associate stream buffers with objects of class `span` as described in [span].

```
namespace std {
namespace experimental {
  template <class charT, class traits = char_traits<charT> >
    class basic_spanbuf;
  typedef basic_spanbuf<char>     spanbuf;
  typedef basic_spanbuf<wchar_t> wspanbuf;
  template <class charT, class traits = char_traits<charT> >
    class basic_ispanstream;
  typedef basic_ispanstream<char>     ispanstream;
  typedef basic_ispanstream<wchar_t> wispanstream;
  template <class charT, class traits = char_traits<charT> >
    class basic_ospanstream;
  typedef basic_ospanstream<char>     ospanstream;
  typedef basic_ospanstream<wchar_t> wospanstream;
  template <class charT, class traits = char_traits<charT> >
    class basic_spanstream;
  typedef basic_spanstream<char>     spanstream;
  typedef basic_spanstream<wchar_t> wspanstream;
}}
```

7.2 27.x `basic_spanbuf` [spanbuf]

TODO

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.

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

explicit basic_spanbuf(
        const basic_span<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_spanbuf<charT, traits, Allocator>`, the type `allocator_-traits<Allocator>::value_type` shall name the same type as `charT`. Every object of type `basic_spanbuf<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 `span<charT>` internally, will simply pass the allocator parameter to the corresponding `span<charT>` constructors. — *end note*]

7.2.1 27.8.2.1 `basic_spanbuf` constructors [spanbuf.cons]

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

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

and

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

7.3 27.8.3 Adjust synopsis of basic_ispanstream [ispanstream]

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_ispanstream(
              ios_base::openmode which = ios_base::in,
              const Allocator &a=Allocator());
  explicit basic_ispanstream(
              const span<charT> <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_ispanstream<charT, traits, Allocator>`, the type `allocator_-traits<Allocator>::value_type` shall name the same type as `charT`. Every object of type `basic_ispanstream<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 `span<charT>` internally, will simply pass the allocator parameter to the corresponding `span<charT>` constructors. — *end note*]

7.3.1 27.8.3.1 `basic_ispanstream` constructors [ispanstream.cons]

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

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

1 *Effects:* Constructs an object of class `basic_ispanstream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_-spanbuf<charT, traits, Allocator>(which | ios_base::in, a))` (27.8.2.1).

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

2 *Effects:* Constructs an object of class `basic_ispanstream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_-spanbuf<charT, traits, Allocator>(str, which | ios_base::in, a))` (27.8.2.1).

7.4 27.8.4 Adjust synopsis of `basic_ospanstream` [ospanstream]

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_ospanstream(
             ios_base::openmode which = ios_base::out,
             const Allocator &a=Allocator());
explicit basic_ospanstream(
             const span<charT> <charT, traits, Allocator>& str,
             ios_base::openmode which = ios_base::out,
             const Allocator &a=Allocator());
```

Append a paragraph p2 to the text following the synopsis:

1 In every specialization `basic_ospanstream<charT, traits, Allocator>`, the type `allocator_-traits<Allocator>::value_type` shall name the same type as `charT`. Every object of type `basic_ospanstream<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 `span<charT>` internally, will simply pass the allocator parameter to the corresponding `span<charT>` constructors. — *end note*]

7.4.1 27.8.4.1 `basic_ospanstream` constructors [ospanstream.cons]

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

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

1 *Effects:* Constructs an object of class `basic_ospanstream`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_spanbuf<charT, traits, Allocator>(which | ios_base::out, a))` (27.8.2.1).

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

2 *Effects:* Constructs an object of class `basic_ospanstream<charT, traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_spanbuf<charT, traits, Allocator>(str, which | ios_base::out, a))` (27.8.2.1).

7.5 27.8.5 Adjust synopsis of `basic_spanstream` [spanstream]

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_spanstream(
             ios_base::openmode which = ios_base::out | ios_base::in,
             const Allocator &a=Allocator());
explicit basic_ospanstream(
             const span<charT> <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:

1 In every specialization `basic_spanstream<charT, traits, Allocator>`, the type `allocator_traits<Allocator>::value_type` shall name the same type as `charT`. Every object of type `basic_spanstream<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 `span<charT>` internally, will simply pass the allocator parameter to the corresponding `span<charT>` constructors. *— end note*]

7.5.1 27.8.5.1 `basic_spanstream` constructors [spanstream.cons]

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

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

1 *Effects:* Constructs an object of class `basic_spanstream<charT, traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_spanbuf<charT, traits, Allocator>(which, a)`.

```
explicit basic_spanstream(
  const basic_span<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_spanstream<charT, traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_spanbuf<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 `span<charT>` .