Formatters for library types

Document #:	P1636R0
Date:	2019-06-14
Project:	Programming Language C++ Library Evolution

Reply-to:	Lars Gullik Bjønnes <lbjonnes@cisco.com>

1 Introduction

After [P0645] and [P1361] almost all types that have a output stream `operator<<` overload will also have a `std::formatter` specialization. The following types, which have output stream operators, are missing this specialization:

- `basic_streambuf`
- `bitset`
- `complex`
- `error_code`
- `filesystem::path`
- `shared_ptr`
- `sub_match`
- `thread::id`
- `unique_ptr`

Adding formatter specializations for all or most of these will reduce surprises for users that convert from stream centric output, to format centric output:

Before

```
std::filesystem::path p;
os << p; // OK
std::format("{}", p); // error
std::format("{}", p.string()); // OK
```

After

```
std::filesystem::path p;
os << p; // OK
std::format("{}", p); // OK
std::format("{}", p.string()); // OK
```

And similar for most types, but also:

Before

```
std::complex<double> c;
os << c; // OK
std::format("{}", c); // error
std::format("({},{})",
            c.real(), c.imag()); // OK
```

After

```
std::complex<double> c;
os << c; // OK
std::format("{}", c); // OK
std::format("({},{})",
            c.real(), c.imag()); // OK
```

1.1 Some considerations:

`bitset operator<<` uses locale, for that reason this proposal suggest `b.template to_string<charT>()` as the wanted output instead.

`unique_ptr` and `shared_ptr` use `.get()` in `operator<<`, so one option is to not add formatters for those since `format` formatters are disabled for pointer types in general. The proposal adds formatters for the smart-pointers, but where the result from `.get()` is cast to `void*` before output.

This proposal does not suggest to use `ostream` to format `complex`, since stream output of `complex` takes locale into account which is probably do not wanted. Also using stream rules to output the `.imag()` and `.real()` parts does not follow how `format` outputs floats. `complex{1.0, 2.0}`, output with streams gives "`(1,2)`", whereas `format("({},{})", c.imag(), c.real()` gives "`(1.0,2.0)`". This proposal specifies a custom `format-spec` that follows the `std-format-spec` closely, where sign, alternate representation, precision and type applies to the inner T, and where fill, alignment and width applies to the value as a whole. The alignment option '=' and preceding the width field with zero ('0') is not allowed.

Leave the `basic_streambuf` alone and let ostream handle that.

1.2 Proposal

Add `formatter` specializations for:

- `error_code`
- `bitset`
- `unique_ptr`
- `shared_ptr`
- `complex`
- `filesystem::path`
- `sub_match`
- `thread::id`

1.2.1 Design questions

OK with using format string specifications for the underlying type:

- error_code -> string
- fs::path -> string
- bitset -> string
- etc. ?

Want format of complex to follow `format("{}", 2.0)` output and not `cout << 2.0` output?

Add the formatters for `unique_ptr` and `shared_ptr` even if formatting of pointers are normally disabled in `std::format`.

1.2.2 Specification questions

Inherit from the *underlying* type to get the format specification implicit from there, or drop the inheritance and put requirement on format specification in the normative test and leave it to the implementation how to get the format specification done?

Use `auto` as return type from formatter functions instead of the verbose `typename basic_format_context<Out, charT>::iterator`?

1.3 Impact on the standard

This proposal is a pure library extension.

1.4 Implementation

All of the formatter specializations in this paper has been implemented on top of [{fmt}].

2 Proposed Wording

Wording is relative to [N4810], with the assumption that [P0645] and [P1361] have also been added.

2.1 error_code

Add to 19.5.1 Header `<system_error>` synopsis [system.error.syn] just below `operator<<`:

```
// 19.5.3.6, formatter
template<class charT> struct formatter<error_code, charT>;
```

Add a new section 19.5.3.6 Formatter [syserr.errcode.fmt]

19.5.3.6 Formatter **[syserr.errcode.fmt]**

```
template<class charT>
  struct formatter<error_code, charT> : formatter<basic_string<charT>, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(error_code ec, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(error_code ec, basic_format_context<Out, charT>& ctx);
```

1 *Effects:* As if implemented by:

```
basic_ostringstream<charT> o;
o.imbue(locale::classic());
o << ec;
return formatter<basic_string<charT>, charT>::format(o.str(), ctx);
```

2.2 bitset

Add to 20.9.1 Header `<bitset>` synopsis [bitset.syn], just below `operator<<`:

// 20.9.5, *bitset formatter*

```
template<size_t N, class charT> struct formatter<bitset<N>, charT>;
```

Add a new section: 20.9.5 bitset formatter [bitset.fmt]

20.9.5 bitset formatter **[bitset.fmt]**

```
template<size_t N, class charT>
  struct formatter<bitset<N>, charT> : formatter<basic_string<charT>, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const bitset<N>& b, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(const bitset<N>& b, basic_format_context<Out, charT>& ctx);
```

1 *Effects:* As if implemented by:

```
return formatter<basic_string<charT>, charT>::format(b.template to_string<charT>(), ctx);
```

2.3 unique_ptr

Add to 20.10.2 Header `<memory>` synopsis [memory.syn] just after `operator<<`:

```
template<class T, class D, class charT> struct formatter<unique_ptr<T, D>, charT>;
```

Add a new section 20.11.1.7 Formatter [unique.ptr.fmt]

20.11.1.7 Formatter **[unique.ptr.fmt]**

```
template<class T, class D, class charT>
  struct formatter<unique_ptr<T, D>, charT> : formatter<void*, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const unique_ptr<T>& p, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(const unique_ptr<T>& p, basic_format_context<Out, charT>& ctx);
```

1 *Constraints:* `p.get()` to be a valid expression.

2 *Effects:* As if implemented by:

```
return formatter<void*, charT>::format(static_cast<void*>(p.get()), ctx);
```

2.4 shared_ptr

Add to 20.10.2 Header `<memory>` synopsis [memory.syn] just after `operator<<`:

```
template<class T, class charT> struct formatter<shared_ptr<T>, charT>;
```

Add a new section 20.11.3.12 Formatter [util.smartptr.shared.fmt]

20.11.3.12 Formatter **[util.smartptr.shared.fmt]**

```
template<class T, class charT>
  struct formatter<shared_ptr<T>, charT> : formatter<void*, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const shared_ptr<T>& p, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(const shared_ptr<T>& p, basic_format_context<Out, charT>& ctx);
```

1 *Constraints:* `p.get()` to be a valid expression.

2 *Effects:* As if implemented by:

```
return formatter<void*, charT>::format(static_cast<void*>(p.get()), ctx);
```

2.5 complex

Add to 26.4.1 Header `<complex>` synopsis [complex.syn] just below `operator<<`:

```
// 26.4.?, formatter
template<class T, class charT> struct formatter<complex<T>, charT>;
```

Add a new section 26.4.? Formatter [complex.fmt]:

26.4.? Formatter **[complex.fmt]**

```
template<class T, class charT>
  struct formatter<complex<T>, charT> {
    typename basic_format_parse_context<charT>::iterator
      parse(basic_format_parse_context& ctx);
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const complex<T>& c, basic_format_context<Out, charT>& ctx);
};

typename basic_format_parse_context::iterator
  parse(basic_format_parse_context&);
```

1 *Effects:* Parses the `format-spec` as per the `std-format-spec` rules for `T`, except:

— right alignment ('>') as default as for arithmetic types
— '=' is removed from the allowed alignment options
— preceding the width field with zero ('0') is not allowed

```
template<class Out>
  typename basic_format_context<Out, charT>::iterator
   format(const complex<T>& c, basic_format_context<Out, charT>& ctx);
```

2 *Effects:* As if implemented by:

```
// pseudo-code
std::basic_string<charT> s =
    std::format("({:[sign] ['#'] ['.' precision] [type]},"
                "{:[sign] ['#'] ['.' precision] [type]})",
                 c.real(), c.imag());
return formatter<basic_string<charT>, charT>::format(
    std::format("{:[[fill] align][width]}", s, ctx);
```

2.6 filesystem::path

Add to 29.11.5 Header `<filesystem>` synopsis [fs.filesystem.syn] (location up to editors discretion):

```
// 29.11.7.8, formatter
template<class charT> struct formatter<filesystem::path, charT>;
```

Add a new section 29.11.7.8 Formatter [fs.path.fmt]

29.11.7.8 Formatter **[fs.path.fmt]**

```
template<class charT>
  struct formatter<filesystem::path, charT> : formatter<basic_string<charT>, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const filesystem::path& p, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(const filesystem::path& p, basic_format_context<Out, charT>& ctx);
```

1 *Effects:* As if implemented by:

```
basic_ostringstream<charT> o;
o.imbue(locale::classic());
o << p;
return formatter<basic_string<charT>, charT>::format(o.str(), ctx);
```

2.7 sub_match

Add to 30.4 Header `<regex>` synopsis [re.syn] after the `operator<<`:

```
// 30.9.3, formatter
template<class BiIter, class charT> struct formatter<sub_match<BiIter>, charT>;
```

Add a new section 30.9.3 Formatter [re.submatch.fmt]

30.8.3 Formatter **[re.submatch.fmt]**

```
template<class BiIter, class charT>
  struct formatter<sub_match<BiIter>, charT> : formatter<basic_string<charT>, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(const sub_match<BiIter>& s, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format(const sub_match<BiIter>& s, basic_format_context<Out, charT>& ctx);
```

1 *Constraints:* `is_same<sub_match<BiIter>::value_type, charT>`

2 *Effects:* As if implemented by:

```
return formatter<basic_string<charT>, charT>::format(s.str(), ctx);
```

2.8 thread::id

Add to 32.3.2.1 Class `thread::id` [thread.thread.id] just after `operator<<`:

```
template<class charT> struct formatter<thread:id, charT>;
```

Add a new paragraph 14 just after paragraph 13:

```
template<class charT>
  struct formatter<thread::id, charT> : formatter<basic_string<charT>, charT> {
    template<class Out>
      typename basic_format_context<Out, charT>::iterator
        format(thread::id id, basic_format_context<Out, charT>& ctx);
};

template<class Out>
  typename basic_format_context<Out, charT>::iterator
    format<thread::id id, basic_format_context<Out, charT>& ctx);
```

14 *Effects:* As if implemented by:

```
basic_ostringstream<charT> o;
o.imbue(locale::classic());
o << id;
return formatter<basic_string<charT>, charT>::format(o.str(), ctx);
```

3 Acknowledgements

Victor Zverovich, Jonathan Wakely and Juan Alday for looking through the draft and giving valuable comments and directions.

4 References

[{fmt}] Zverovich Victor. A modern formatting library.
https://github.com/fmtlib/fmt

[N4810] Richard Smith. Working draft, Standard for Programming Language C++.
http://wg21.link/N4810

[P0645] Victor Zverovich. Text Formatting.
http://wg21.link/P0645

[P1361] Victor Zverovich, et al. Integration of chrono with text formatting.
http://wg21.link/P1361