Integration of chrono with text formatting

Document #:	P1361R0
Date:	2018-10-16
Project:	Programming Language C++ Library Evolution Working Group, Library Working Group
Reply-to:	Victor Zverovich <victor.zverovich@gmail.com>

"If fmt (P0645) moves forward within the LEWG, this section (Formatting) can easily be reworked to plug into that facility without loss of functionality. This will avoid two unrelated format facilities in the standard."

– [P0355]

1 Motivation

[P0355] that includes a `strftime`-like formatting facility for chrono types was adopted into the draft standard for C++20 in Jacksonville. Meanwhile [P0645] that provides a more general formatting facility was accepted by the Library Evolution working group in San Diego and forwarded to the Library working group for a wording review also targeting C++20. In this paper we propose revising the output APIs added by [P0355] based on [P0645].

Integrating the two proposals provides the following advantages:

1. Easier formatting of multiple objects and positional arguments support:

   Before

   ```
   void print_birthday(std::string_view name,
                       const std::chrono::year_month_day& birthday) {
     std::cout << name << "'s birthday is "
               << std::chrono::format("%Y-%m-%d", birthday) << ".\n";
   }
   ```

   After

   ```
   void print_birthday(std::string_view name,
                       const std::chrono::year_month_day& birthday) {
     std::cout << std::format("{0}'s birthday is {1:%Y-%m-%d}.\n", name, birthday);
   }
   ```

2. Output iterator support and the ability to easily avoid dynamic memory allocations:

   Before

   ```
   std::string str = std::chrono::format("%Y-%m-%d", date);
   ```

   After

   ```
   std::array<char, 100> buf;
   std::format_to_n(buf.data(), buf.size(), "{:%Y-%m-%d}", date);
   ```

3. Prevent confusing overload resolution:

   Before

   ```
   std::chrono::year_month_day date;
   format("...", date);                      // resolves to std::chrono::format
   format(std::string_view("..."), date); // resolves to std::format
   ```

   After

   ```
   std::chrono::year_month_day date;
   format("...", date);                      // resolves to std::format
   format(std::string_view("..."), date); // resolves to std::format
   ```

4. Allow fill, width, and alignment in a format string using the same syntax as for other types:

   Before

   ```
   std::cout << std::setw(20) << std::right
             << std::chrono::format("%Y-%m-%d", birthday) << "\n";
   ```

   After

   ```
   std::cout << std::format("{0:>20%Y-%m-%d}\n", birthday);
   ```

2 Locale

One feature that [P0355] has and [P0645] doesn't is the ability to pass a locale to a formatting fuction. We propose extending the format API of P0645 to allow the same.

Before

```
auto zt = std::chrono::zoned_time(...);
std::cout << "Localized time is "
          << std::chrono::format(std::locale{"fi_FI"}, "%c", zt) << "\n";
```

After

```
auto zt = std::chrono::zoned_time(...);
std::cout << std::format(std::locale{"fi_FI"}, "Localized time is {:%c}\n", zt);
```

3 Open Questions

HH There's also a std::chrono::parse which this doesn't have. For symmetry purposes I think we need to alias this to std::chrono as well, but I have not tried that. Do we care about the lack of symmetry?

4 Proposed Wording

This wording is based on the working draft [N4727] unless stated otherwise.

Modify section 25.2 Header `<chrono>` synopsis [time.syn]:

```
  // 25.5.10, duration I/O
  template<class charT, class traits, class Rep, class Period>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os,
                 const duration<Rep, Period>& d);
- template<class charT, class traits, class Rep, class Period>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const duration<Rep, Period>& d);
+ template<class Rep, class Period, class charT>
+   struct formatter<duration<Rep, Period>, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const sys_days& dp);

- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const sys_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<sys_time<Duration>, charT>;

...

  template<class charT, class traits, class Duration>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const utc_time<Duration>& t);
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const utc_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<utc_time<Duration>, charT>;

...

  template<class charT, class traits, class Duration>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const tai_time<Duration>& t);
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const tai_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<tai_time<Duration>, charT>;

...
```

```
  template<class charT, class traits, class Duration>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const gps_time<Duration>& t);
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const gps_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<gps_time<Duration>, charT>;

...

template<class charT, class traits, class Duration>
  basic_ostream<charT, traits>&
    operator<<(basic_ostream<charT, traits>& os, const file_time<Duration>& tp);
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const file_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<file_time<Duration>, charT>;

...

  template<class charT, class traits, class Duration>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const local_time<Duration>& tp);
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const local_time<Duration>& tp,
-               const string* abbrev = nullptr, const seconds* offset_sec = nullptr);
+ template<class Duration, class charT>
+   struct formatter<local_time<Duration>, charT>;
+
+ template<class Duration> struct local_time_format;
+ template<class Duration, class charT>
+   struct formatter<local_time_format<Duration>, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const day& d);
- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const day& d);
+ template<class charT>
+   struct formatter<day, charT>;

...
```

```
  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const month& m);
- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const month& m);
+ template<class charT>
+   struct formatter<month, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year& y);

- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const year& y);
+ template<class charT>
+   struct formatter<year, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const weekday& wd);

- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const weekday& wd);
+ template<class charT>
+   struct formatter<weekday, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const weekday_indexed& wdi);
+ template<class charT>
+   struct formatter<weekday_indexed, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const weekday_last& wdl);
+ template<class charT>
+   struct formatter<weekday_last, charT>;

...
```

```
  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const month_day& md);
- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const month_day& md);
+ template<class charT>
+   struct formatter<month_day, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
       operator<<(basic_ostream<charT, traits>& os, const month_day_last& mdl);
+ template<class charT>
+   struct formatter<month_day_last, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const month_weekday& mwd);
+ template<class charT>
+   struct formatter<month_weekday, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const month_weekday_last& mwdl);
+ template<class charT>
+   struct formatter<month_weekday_last, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year_month& ym);

- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const year_month& ym);
+ template<class charT>
+   struct formatter<year_month, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year_month_day& ymd);
```

```
- template<class charT, class traits>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const year_month_day& ymd);
+ template<class charT>
+   struct formatter<year_month_day, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year_month_day_last& ymdl);
+ template<class charT>
+   struct formatter<year_month_day_last, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year_month_weekday& ymwdi);
+ template<class charT>
+   struct formatter<year_month_weekday, charT>;

...

  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const year_month_weekday_last& ymwdl);
+ template<class charT>
+   struct formatter<year_month_weekday_last, charT>;

...

  template<class charT, class traits, class Rep, class Period>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os,
                 const time_of_day<duration<Rep, Period>>& t);
+ template<class Rep, class Period, class charT>
+   struct formatter<time_of_day<duration<Rep, Period>>, charT>;

...

  // 25.10.4, information classes
  struct sys_info;
  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const sys_info& si);
+ templateclass charT>
+   struct formatter<sys_info, charT>;
```

```
  struct local_info;
  template<class charT, class traits>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const local_info& li);
+ template<class charT>
+   struct formatter<local_info, charT>;

...

  template<class charT, class traits, class Duration, class TimeZonePtr>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os,
                 const zoned_time<Duration, TimeZonePtr>& t);

- template<class charT, class traits, class Duration, class TimeZonePtr>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const zoned_time<Duration, TimeZonePtr>& tp);
+ template<class Duration, class TimeZonePtr, class charT>
+   struct formatter<zoned_time<Duration, TimeZonePtr>, charT>;

...

- // 25.11, formatting
- template<class charT, class Streamable>
-   basic_string<charT>
-     format(const charT* fmt, const Streamable& s);
- template<class charT, class Streamable>
-   basic_string<charT>
-     format(const locale& loc, const charT* fmt, const Streamable& s);
- template<class charT, class traits, class Alloc, class Streamable>
-   basic_string<charT, traits, Alloc>
-     format(const basic_string<charT, traits, Alloc>& fmt, const Streamable& s);
- template<class charT, class traits, class Alloc, class Streamable>
-   basic_string<charT, traits, Alloc>
-     format(const locale& loc, const basic_string<charT, traits, Alloc>& fmt,
-            const Streamable& s);
```

Modify section 25.5.10 I/O [time.duration.io]:

```
- template<class charT, class traits, class Rep, class Period>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt,
-               const duration<Rep, Period>& d);
+ template<class Rep, class Period, class charT>
+   struct formatter<duration<Rep, Period>, charT>;
```

6 ~~*Effects:* Streams `d` into `os` using the format specified by the NTCTS `fmt`. `fmt` encoding follows the rules specified in 25.11.~~

7 ~~*Returns:* `os`.~~

The specialization is enabled and the format string encoding follows the rules specified in 25.11.

Modify section 25.7.1.3 Non-member functions [time.clock.system.nonmembers]:

```
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const sys_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<sys_time<Duration>, charT>;
```

7 *Effects:* Streams tp into os using the format specified by the NTCTS fmt. fmt encoding follows the rules specified in 25.11. If %Z is used, it will be replaced with "UTC" widened to charT. If %z is used (or a modified variant of %z), an offset of 0min will be formatted.

8 *Returns:* os.

The specialization is enabled and the format string encoding follows the rules specified in 25.11. If %Z is used, it will be replaced with "UTC" widened to charT. If %z is used (or a modified variant of %z), an offset of 0min will be formatted.

Modify section 25.7.2.3 Non-member functions [time.clock.utc.nonmembers]:

1 *Effects:* Calls to_stream(os, fmt, t), where fmt is a string containing "%F %T" widened to charT.

1 *Effects:* Equivalent to os << format(fmt, t), where fmt is a string containing "{:%F %T}" widened to charT.

```
- template<class charT, class traits, class Duration>
-   basic_ostream<charT, traits>&
-     to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const utc_time<Duration>& tp);
+ template<class Duration, class charT>
+   struct formatter<utc_time<Duration>, charT>;
```

3 *Effects:* Streams tp into os using the format specified by the NTCTS fmt. fmt encoding follows the rules specified in 25.11. If %Z is used, it will be replaced with "UTC" widened to charT. If %z is used (or a modified variant of %z), an offset of 0min will be formatted. If tp represents a time during a leap second insertion, and if a seconds field is formatted, the integral portion of that format shall be "60" widened to charT.

4 *Returns:* os.

The specialization is enabled and the format string encoding follows the rules specified in 25.11. If %Z is used, it will be replaced with "UTC" widened to charT. If %z is used (or a modified variant of %z), an offset of 0min will be formatted. If tp represents a time during a leap second insertion, and if a seconds field is formatted, the integral portion of that format shall be "60" widened to charT.

Modify section 25.7.3.3 Non-member functions time.clock.tai.nonmembers:

1 *Effects:* Calls to_stream(os, fmt, t), where fmt is a string containing "%F %T" widened to charT.

1 *Effects:* Equivalent to os << format(fmt, t), where fmt is a string containing "{:%F %T}" widened to charT.

TODO: replace all to_stream overloads with formatter specializations and rephrase references to to_stream in terms of format.

Modify section 25.11 Formatting [time.format]:

1 Each format overload specified in this subclause calls to_stream unqualified, so as to enable argument dependent lookup (6.4.2).

```
template<class charT, class Streamable>
  basic_string<charT>
    format(const charT* fmt, const Streamable& s);
```

...

13 Returns: `os.str()`.

14 The `format` functions call a `to_stream` function with a `basic_ostream`, a formatting string specifier, and a `Streamable` argument. Each `to_stream` overload is customized for each `Streamable` type. However all `to_stream` overloads treat the formatting string specifier according to the following specification:

15 The `fmt` string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a `%` character, possibly followed by an `E` or `O` modifier character (described below), followed by a character that determines the behavior of the conversion specifier. All ordinary multibyte characters (excluding the terminating null character) are streamed unchanged into the `basic_ostream`.

Each `formatter` specialization in the chrono library (25.2) satisfies the *Formatter* requirements ([formatter.requirements]).

The `parse` member functions of these formatters treat the formatting string according to the following specification:

```
format-spec     ::= [[fill] align] [width] [conversion-spec [chrono-specs]]
chrono-specs    ::= chrono-spec [chrono-specs]
chrono-spec     ::= literal-char | conversion-spec
literal-char    ::= <a character other than '{' or '}'>
conversion-spec ::= '%' [modifier] type
modifier        ::= 'E' | 'O'
type            ::= 'a' | 'A' | 'b' | 'B' | 'c' | 'C' | 'd' | 'D' | 'e' | 'F' | 'g' |
                    'G' | 'h' | 'H' | 'I' | 'j' | 'm' | 'M' | 'n' | 'p' | 'r' | 'R' |
                    'S' | 't' | 'T' | 'u' | 'U' | 'V' | 'w' | 'W' | 'x' | 'X' | 'y' |
                    'Y' | 'z' | 'Z' | '%'
```

`fill`, `align`, and `width` are described in Section [format.functions]. All ordinary multibyte characters represented by `literal-char` are copied unchanged to the output.

16 Each conversion specifier is replaced by appropriate characters as described in Table 88. Some of the conversion specifiers depend on the locale which is imbued to the `basic_ostream`. If the `Streamable` object does not contain the information the conversion specifier refers to, the value streamed to the `basic_ostream` is unspecified.

Each conversion specifier `conversion-spec` is replaced by appropriate characters as described in Table 88. Some of the conversion specifiers depend on the locale which is passed to the formatting function if the latter takes one or the global locale otherwise. If the formatted object does not contain the information the conversion specifier refers to, the value written to the output is unspecified.

17 Unless explicitly specified, `Streamable` types will not contain time zone abbreviation and time zone offset information. If available, the conversion specifiers `%Z` and `%z` will format this information (respectively). If the information is not available, and `%Z` or `%z` are contained in `fmt`, `os.setstate(ios_base::failbit)` shall be called.

Unless explicitly specified, formatted chrono types will not contain time zone abbreviation and time zone offset information. If available, the conversion specifiers `%Z` and `%z` will format this information (respectively). If the information is not available, and `%Z` or `%z` are contained in `format-spec`, `format_error` shall be thrown.

Table 88 – Meaning of `format` conversion specifiers

Specifier	Replacement
`%a`	The locale's abbreviated weekday name. If the value does not contain a valid weekday, `setstate(ios::failbit)` is called `format_error` is thrown.
`%A`	The locale's full weekday name. If the value does not contain a valid weekday, `setstate(ios::failbit)` is called `format_error` is thrown.
`%b`	The locale's abbreviated month name. If the value does not contain a valid month, `setstate(ios::failbit)` is called `format_error` is thrown.
TODO	remaining conversion specifiers

If the format specification contains no conversion specifiers then the chrono object is formatted as if by streaming it to `std::ostringstream os` and copying `os.str()` through the output iterator of the context with additional padding and adjustments as per format specifiers.

[*Example:*

```
string s = format("{:>8}", 42ms); // s == "    42ms"
```

— *end example*]

TODO: global or C locale?

4.1 Changes to P0645 Text Formatting

The wording in this section is based on [P0645].

Modify section 19.20.1 Header `<format>` synopsis [format.syn]:

```
  template<class... Args>
    wstring format(wstring_view fmt, const Args&... args);
+ template<class... Args>
+   string format(const locale& loc, string_view fmt, const Args&... args);
+ template<class... Args>
+   wstring format(const locale& loc, wstring_view fmt, const Args&... args);

...

  wstring vformat(wstring_view fmt, wformat_args args);
+ string vformat(const locale& loc, string_view fmt, format_args args);
+ wstring vformat(const locale& loc, wstring_view fmt, wformat_args args);

...

  template<OutputIterator<const wchar_t&> O, class... Args>
    O format_to(O out, wstring_view fmt, const Args&... args);
+ template<OutputIterator<const char&> O, class... Args>
+   O format_to(O out, const locale& loc, string_view fmt, const Args&... args);
+ template<OutputIterator<const wchar_t&> O, class... Args>
+   O format_to(O out, const locale& loc, wstring_view fmt, const Args&... args);

...

  template<OutputIterator<const wchar_t&> O>
```

```
    O vformat_to(O out, wstring_view fmt, format_args_t<O, wchar_t> args);
+ template<OutputIterator<const char&> O>
+   O vformat_to(O out, const locale& loc, string_view fmt,
+                format_args_t<O, char> args);
+ template<OutputIterator<const wchar_t&> O>
+   O vformat_to(O out, const locale& loc, wstring_view fmt,
+                format_args_t<O, wchar_t> args);

...

  template<OutputIterator<const char&> O, class... Args>
    format_to_n_result<O> format_to_n(O out, iter_difference_t<O> n,
                                      string_view fmt, const Args&... args);
+ template<OutputIterator<const char&> O, class... Args>
+   format_to_n_result<O> format_to_n(O out, iter_difference_t<O> n,
+                                     const locale& loc, string_view fmt,
+                                     const Args&... args);
+ template<OutputIterator<const wchar_t&> O, class... Args>
+   format_to_n_result<O> format_to_n(O out, iter_difference_t<O> n,
+                                     const locale& loc, wstring_view fmt,
+                                     const Args&... args);

...

  template<class... Args>
    size_t formatted_size(wstring_view fmt, const Args&... args);
+ template<class... Args>
+   size_t formatted_size(const locale& loc, string_view fmt,
+                         const Args&... args);
+ template<class... Args>
+   size_t formatted_size(const locale& loc, wstring_view fmt,
+                         const Args&... args);
```

Modify section 19.20.2 Formatting functions [format.functions]:

```
template<class... Args>
  string format(const locale& loc, string_view fmt, const Args&... args);
```

Returns: `vformat(loc, fmt, make_format_args(args...))`.

```
template<class... Args>
  wstring format(const locale& loc, wstring_view fmt, const Args&... args);
```

Returns: `vformat(loc, fmt, make_wformat_args(args...))`.

```
string vformat(const locale& loc, string_view fmt, format_args args);
```

Returns: A string object holding the character representation of formatting arguments provided by args formatted according to specifications given in `fmt` using `loc` for locale-specific formatting.

Throws: `format_error` if `fmt` is not a valid format string.

TODO: remaining functions

Modify section 19.20.3.3 Class template `basic_format_context` [format.context]:

```
  template<class O, class charT> requires OutputIterator<O, const charT&>
    class basic_format_context {
    public:
...
      basic_format_arg<basic_format_context> arg(size_t id) const;
+     std::locale locale();
...
    };
```

`std::locale locale();`

Returns: The locale passed to a formatting function if the latter takes one or `std::locale()` otherwise.

5 References

[N4727] Richard Smith. 2018. Working Draft, Standard for Programming Language C++.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4727.pdf

[P0355] Howard E. Hinnant and Tomasz Kamiński. 2018. Extending to Calendars and Time Zones.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0355r7.html

[P0645] Victor Zverovich. 2018. Text Formatting.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0645r3.html