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

   After

```
std::cout << std::format("{0:>15%Y-%m-%d}\n", birthday);
```

5. Improve control over formatting:

   Before

```
std::cout << std::left << std::setw(8) << Sunday[2] << "game\n";
// prints "Sun     [2]game"
//                ^ note misaligned index and width applying only to
//                  Sunday
```

   After

```
std::cout << std::format("{0:<8}{1}\n", Sunday[2], "game");
// prints "Sun[2]  game"
```

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 Proposed changes

We propose the following changes to [N4727] and [P0645]:

1. Replace `std::chrono::to_stream` overloads with `std::formatter` specializations to make chrono types formattable with functions from [P0645], e.g.

```
namespace chrono {
- 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<chrono::duration<Rep, Period>, charT>;
```

2. Remove `std::chrono::format` in favor of `std::format`, `std::format_to`, and other formatting functions provided by [P0645].

3. Extend format specifications to allow width, fill, and alignment for consistency with specifications for other types:

   ```
   format-spec     ::= [[fill] align] [width] [conversion-spec [chrono-specs]]
   ```

   Example:

   ```
   string s = format("{0:>15%Y-%m-%d}", birthday);
   // s == "     1950-12-30"
   ```

4. Specify that the default format `"{}"` produces the same output as `operator<<`, e.g.

   ```
   string s = format("{}", 10ms);
   // s == "10ms"
   ```

5. Restate `operator<<` definitions in terms of `std::format` to make I/O manipulators apply to whole objects rather than their parts. For example

   ```
   std::cout << std::left << std::setw(8) << Sunday[2] << "game\n";
   ```

   will print “Sun[2] game” instead of “Sun [2]game”.

6. Add [P0645] formatting function overloads that take a locale and make the locale available to custom formatters via format context, e.g.

   ```
   string s = std::format(std::locale{"fi_FI"}, "{:%c}", zt);
   ```

4 Open Questions

Howard Hinnant: 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?

5 Implementation

Formatting of chrono durations and locale support have been implemented in the {fmt} library.

6 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 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 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 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 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 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 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 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, 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, 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, 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, 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, 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, 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, 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);

...

  // 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);
+ namespace chrono {
+   template<class Duration> struct local_time_format_t; // exposition-only
+
+   template<class Duration>
+     local_time_format_t<Duration>
+       local_time_format(local_time<Duration> time, const string* abbrev = nullptr,
+                         const seconds* offset_sec = nullptr);
+ }
+
+ template<class Rep, class Period, class charT>
+   struct formatter<chrono::duration<Rep, Period>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::sys_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::utc_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::tai_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::gps_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::file_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::local_time<Duration>, charT>;
+ template<class Duration, class charT>
+   struct formatter<chrono::local_time_format_t<Duration>, charT>;
+ template<class charT> struct formatter<chrono::day, charT>;
+ template<class charT> struct formatter<chrono::month, charT>;
+ template<class charT> struct formatter<chrono::year, charT>;
+ template<class charT> struct formatter<chrono::weekday, charT>;
+ template<class charT> struct formatter<chrono::weekday_indexed, charT>;
+ template<class charT> struct formatter<chrono::weekday_last, charT>;
```

```
+ template<class charT> struct formatter<chrono::month_day, charT>;
+ template<class charT> struct formatter<chrono::month_day_last, charT>;
+ template<class charT> struct formatter<chrono::month_weekday, charT>;
+ template<class charT> struct formatter<chrono::month_weekday_last, charT>;
+ template<class charT> struct formatter<chrono::year_month, charT>;
+ template<class charT> struct formatter<chrono::year_month_day, charT>;
+ template<class charT> struct formatter<chrono::year_month_day_last, charT>;
+ template<class charT> struct formatter<chrono::year_month_weekday, charT>;
+ template<class charT> struct formatter<chrono::year_month_weekday_last, charT>;
+ template<class Rep, class Period, class charT>
+   struct formatter<chrono::time_of_day<duration<Rep, Period>>, charT>;
+ templateclass charT> struct formatter<chrono::sys_info, charT>;
+ template<class charT> struct formatter<chrono::local_info, charT>;
+ template<class Duration, class TimeZonePtr, class charT>
+   struct formatter<chrono::zoned_time<Duration, TimeZonePtr>, charT>;
```

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);
```

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`.

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

2 *Effects:*

```
  auto const dp = floor<days>(tp);
- os << year_month_day{dp} << ' ' << time_of_day{tp-dp};
+ os << format(os.getloc(), "{} {}", year_month_day{dp}, time_of_day{tp-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);
```

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`.

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);
```

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`.

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`.

```
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);
```

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 `"TAI"`. If `%z` is used (or a modified variant of `%z`), an offset of `0min` will be formatted. The date and time formatted shall be equivalent to that formatted by a `sys_time` initialized with:

```
sys_time<Duration>{tp.time_since_epoch()} -
  (sys_days{1970y/January/1} - sys_days{1958y/January/1})
```

4 *Returns:* `os`.

5 [*Example*:

```
  auto st = sys_days{2000y/January/1};
  auto tt = clock_cast<tai_clock>(st);
- cout << format("%F %T %Z == ", st) << format("%F %T %Z\n", tt);
+ cout << format("{0:%F %T %Z} == {1:%F %T %Z}\n", st, tt);
```

Produces this output:

```
2000-01-01 00:00:00 UTC == 2000-01-01 00:00:32 TAI
```

— *end example*]

Modify section 25.7.4.3 Non-member functions [time.clock.gps.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 gps_time<Duration>& tp);
```

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 `"GPS"`. If `%z` is used (or a modified variant of `%z`), an offset of `0min` will be formatted. The date and time formatted shall be equivalent to that formatted by a `sys_time` initialized with:

```
sys_time<Duration>{tp.time_since_epoch()} +
  (sys_days{1980y/January/Sunday[1]} - sys_days{1970y/January/1})
```

4 *Returns:* `os`.

5 [*Example*:

```
  auto st = sys_days{2000y/January/1};
  auto gt = clock_cast<gps_clock>(st);
- cout << format("%F %T %Z == ", st) << format("%F %T %Z\n", gt);
+ cout << format("{0:%F %T %Z} == {1:%F %T %Z}\n", st, gt);
```

Produces this output:

```
2000-01-01 00:00:00 UTC == 2000-01-01 00:00:13 GPS
```

— *end example*]

Modify section 25.7.5.3 Non-member functions [time.clock.file.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 file_time<Duration>& tp);
```

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. The > date and time formatted shall be equivalent to that formatted by a `sys_time` initialized with `clock_cast<system_clock>(tp)`, or by a `utc_time` initialized with `clock_cast<utc_clock>(tp)`.

4 *Returns:* `os`.

Modify section 25.7.8 Local time [time.clock.local]:

```
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);
```

4 *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 `*abbrev` if `abbrev` is not equal to `nullptr`. If `abbrev` is equal to `nullptr` (and `%Z` is used), `os.setstate(ios_base::failbit)` shall be called. If `%z` is used (or a modified variant of `%z`), it will be formatted with the value of `*offset_sec` if `offset_sec` is not equal to `nullptr`. If `%z` (or a modified variant of `%z`) is used, and `offset_sec` is equal to `nullptr`, then `os.setstate(ios_base::failbit)` shall be called.

4 *Returns:* `os`.

Modify section 25.8.3.3 Non-member functions [time.cal.day.nonmembers]:

7 *Effects:* Inserts `format(fmt, d)` where `fmt` is `"%d""{:%d}"` widened to `charT`. If `!d.ok()`, appends with `" is not a valid day"`.

```
template<class charT, class traits>
  basic_ostream<charT, traits>&
    to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const day& d);
```

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

10 *Returns:* `os`.

Modify section 25.8.4.3 Non-member functions [time.cal.month.nonmembers]:

7 *Effects:* If `m.ok() == true` inserts `format(os.getloc(), fmt, m)` where `fmt` is `"%b""{:%b}"` widened to `charT`. Otherwise inserts `unsigned{m} << is not a valid month"`.

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

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

10 *Returns:* `os`.

Modify section 25.8.5.3 Non-member functions [time.cal.year.nonmembers]:

7 *Effects:* Inserts `format(fmt, y)` where `fmt` is `"%Y""{%Y:}"` widened to `charT`. If `!y.ok()`, appends with `" is not a valid year"`.

```
template<class charT, class traits>
  basic_ostream<charT, traits>&
    to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const year& y):
```

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

10 *Returns:* `os`.

Modify section 25.8.6.3 Non-member functions [time.cal.wd.nonmembers]:

6 *Effects:* If `wd.ok() == true` inserts `format(os.getloc(), fmt, m)` where `fmt` is `"%a""{:%a}"` widened to `charT`. Otherwise inserts `unsigned{m} << is not a valid weekday"`.

```
template<class charT, class traits>
  basic_ostream<charT, traits>&
    to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const weekday& wd);
```

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

9 *Returns:* `os`.

Modify section 25.8.7.3 Non-member functions [time.cal.wdidx.nonmembers]:

2 *Effects:* `os << wdi.weekday() << '[' << wdi.index()`. If `wdi.index()` is in the range `[1, 5]`, appends with `']'`, otherwise appends with `" is not a valid index]"`.

2 *Effects:* Equivalent to

```
os << format(os.getloc(), "{}[{}{}]", wdi.weekday(), i,
             i >= 1 && i <= 5 ? "" : " is not a valid index");
```

where `i` is `wdi.index()`.

Modify section 25.8.8.3 Non-member functions [time.cal.wdlast.nonmembers]:

2 *Returns:* `os << wdl.weekday() << "[last]"`.

2 *Returns:* `os << format(os.getloc(), "{}[last]", wdl.weekday())`.

Modify section 25.8.9.3 Non-member functions [time.cal.md.nonmembers]:

3 *Returns:* `os << md.month() << '/' << md.day()`.

3 *Returns:* `os << format(os.getloc(), "{}/{}", md.month(), md.day())`.

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

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

9 *Returns:* `os`.

Modify section 25.8.10 Class `month_day_last` [time.cal.mdlast]:

9 *Returns:* `os << mdl.month() << "/last"`.

9 *Returns:* `os << format(os.getloc(), "{}/last", md.month())`.

Modify section 25.8.11.3 Non-member functions [time.cal.mwd.nonmembers]:

2 *Returns:* `os << mwd.month() << '/' << mwd.weekday_indexed()`.

2 *Returns:* `os << format(os.getloc(), "{}/{}", mwd.month(), mwd.weekday_indexed())`.

Modify section 25.8.12.3 Non-member functions [time.cal.mwdlast.nonmembers]:

2 *Returns:* `os << mwdl.month() << '/' << mwdl.weekday_last()`.

2 *Returns:* `os << format(os.getloc(), "{}/{}", mwdl.month(), mwdl.weekday_last())`.

Modify section 25.8.13.3 Non-member functions [time.cal.ym.nonmembers]:

10 *Returns:* `os << ym.year() << '/' << ym.month()`.

10 *Returns:* `os << format(os.getloc(), "{}/{}", ym.year(), ym.month())`.

```
template<class charT, class traits>
  basic_ostream<charT, traits>&
    to_stream(basic_ostream<charT, traits>& os, const charT* fmt, const year_month& ym);
```

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

12 *Returns:* `os`.

Modify section 25.8.14.3 Non-member functions [time.cal.ymd.nonmembers]:

11 *Effects:* Inserts `format(fmt, ymd)` where `fmt` is `"%F""{:%F}"` widened to `charT`. If `!ymd.ok()`, appends with `" is not a valid date"`.

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

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

14 *Returns:* `os`.

Modify section 25.8.15.3 Non-member functions [time.cal.ymdlast.nonmembers]:

9 *Returns:* `os << ymdl.year() << '/' << ymdl.month_day_last()`.

9 *Returns:* `os << format(os.getloc(), "{}/{}", ymdl.year(), ymdl.month_day_last())`.

Modify section 25.8.16.3 Non-member functions [time.cal.ymwd.nonmembers]:

8 *Returns:* `os << ymwdi.year() << '/' << ymwdi.month() << '/' << ymwdi.weekday_indexed()`.

8 *Returns:* `os << format(os.getloc(), "{}/{}/{}", ymwdi.year(), ymwdi.month(), ymwdi.weekday_indexed())`.

Modify section 25.8.17.3 Non-member functions [time.cal.ymwdlast.nonmembers]:

8 *Returns:* `os << ymwdl.year() << '/' << ymwdl.month() << '/' << ymwdl.weekday_last(`.

8 *Returns:* `os << format(os.getloc(), "{}/{}/{}", ymwdl.year(), ymwdl.month(), ymwdl.weekday_last())`.

Modify section 25.10.7.4 Non-member functions [time.zone.zonedtime.nonmembers]:

```
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);
```

5 *Effects:* First obtains a `sys_info` via `tp.get_info()` which for exposition purposes will be referred to as `info`. Then calls `to_stream(os, fmt, tp.get_local_time(), &info.abbrev, &info.offset)`.

6 *Returns:* `os`.

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.
`%B`	The locale's full month name. If the value does not contain a valid month, `setstate(ios::failbit)` is called `format_error` is thrown.
...	...
`%z`	The offset from UTC in the ISO 8601 format. For example `-0430` refers to 4 hours 30 minutes behind UTC. If the offset is zero, `+0000` is used. The modified commands `%Ez` and `%Oz` insert a `:` between the hours and minutes: `-04:30`. If the offset information is not available, `setstate(ios_base::failbit)` shall be called `format_error` shall be thrown.
`%Z`	The time zone abbreviation. If the time zone abbreviation is not available, `setstate(ios_base::failbit)` shall be called `format_error` shall be thrown.
`%%`	A `%` character.

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*]

```
template<class Duration, class charT>
  struct formatter<chrono::sys_time<Duration>, charT>;
```

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.

```
template<class Duration, class charT>
  struct formatter<chrono::utc_time<Duration>, charT>;
```

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.

```
template<class Duration, class charT>
  struct formatter<chrono::tai_time<Duration>, charT>;
```

If %Z is used, it will be replaced with "TAI". If %z is used (or a modified variant of %z), an offset of 0min will be formatted. The date and time formatted shall be equivalent to that formatted by a sys_time initialized with:

```
sys_time<Duration>{tp.time_since_epoch()} -
  (sys_days{1970y/January/1} - sys_days{1958y/January/1})
```

```
template<class Duration, class charT>
  struct formatter<chrono::gps_time<Duration>, charT>;
```

If %Z is used, it will be replaced with "GPS". If %z is used (or a modified variant of %z), an offset of 0min will be formatted. The date and time formatted shall be equivalent to that formatted by a sys_time initialized with:

```
sys_time<Duration>{tp.time_since_epoch()} +
  (sys_days{1980y/January/Sunday[1]} - sys_days{1970y/January/1})
```

```
template<class Duration, class charT>
  struct formatter<chrono::file_time<Duration>, charT>;
```

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. The > date and time formatted shall be equivalent to that formatted by a sys_time initialized with clock_cast<system_clock>(tp), or by a utc_time initialized with clock_cast<utc_clock>(tp).

```
template<class Duration, class charT>
  struct formatter<chrono::local_time<Duration>, charT>;
```

If %Z, %z, or a modified version of %z is used, format_error shall be thrown.

```
template<class Duration> struct local_time_format_t { // exposition-only
  local_time<Duration> time;
  const string* abbrev;
  const seconds* offset_sec;
};

template<class Duration>
  local_time_format_t<Duration>
    local_time_format(local_time<Duration> time, const string* abbrev = nullptr,
                      const seconds* offset_sec = nullptr);
```

Returns: {time, abbrev, offset_sec}.

```
template<class Duration, class charT>
  struct formatter<chrono::local_time_format_t<Duration>, charT>;
```

Let `f` be a `local_time_format_t<Duration>` object passed to `formatter::format`. If `%Z` is used, it will be replaced with `*f.abbrev` if `f.abbrev` is not equal to `nullptr`. If `f.abbrev` is equal to `nullptr` (and `%Z` is used), `format_error` shall be thrown. If `%z` is used (or a modified variant of `%z`), it will be formatted with the value of `*f.offset_sec` if `f.offset_sec` is not equal to `nullptr`. If `%z` (or a modified variant of `%z`) is used, and `f.offset_sec` is equal to `nullptr`, then `format_error` shall be thrown.

```
template<class Duration, class TimeZonePtr, class charT>
  struct formatter<chrono::zoned_time<Duration, TimeZonePtr>, charT>
    : formatter<chrono::local_time_format_t<Duration>, charT> {
  template <typename FormatContext>
    typename FormatContext::iterator
      format(const chrono::zoned_time<Duration, TimeZonePtr>& tp, FormatContext& ctx);
};

template <typename FormatContext>
  typename FormatContext::iterator
    format(const chrono::zoned_time<Duration, TimeZonePtr>& tp, FormatContext& ctx);
```

Effects: First obtains a `sys_info` via `tp.get_info()` which for exposition purposes will be referred to as `info`. Then returns `formatter<chrono::local_time_format_t<Duration>, charT>::format( {tp.get_local_time(), &info.abbrev, &info.offset}, ctx)`.

6.1 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);`
`wstring vformat(const locale& loc, wstring_view fmt, wformat_args args);`

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

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

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

Returns: `vformat_to(out, loc, fmt, make_format_args<basic_format_context<O, char>>(args...))`.

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

Returns: `vformat_to(out, loc, fmt, make_format_args<basic_format_context<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);
```

Effects: Places the character representation of formatting arguments provided by `args` formatted according to specifications given in `fmt` into the range `[out, out + N)`, where `N` is the formatted output size. Uses `loc` for locale-specific formatting.

Returns: `out + N`.

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

```
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);
```

Let `N` be the formatted output size and `M` be `min(max(n, 0), N)`.

Effects: Places the character representation of formatting arguments provided by `args` formatted according to specifications given in `fmt` into the range `[out, out + M)`. Uses `loc` for locale-specific formatting.

Returns: `{out + M, N}`.

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

```
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);
```

Returns: The number of characters in the character representation of formatting arguments `args` formatted according to specifications given in `fmt`. Uses `loc` for locale-specific formatting.

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

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.

7 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