Fixing locale handling in chrono formatters

D2372R1

Victor Zverovich (victor.zverovich@gmail.com)

Corentin Jabot (corentin.jabot@gmail.com)

The problem

- In C++20 "Extending `<chrono>` to Calendars and Time Zones" ([P0355]) and "Text Formatting" ([P0645]) proposals were integrated ([P1361]).
- Missed a design issue: `std::format` is locale-independent by default and provides control over locale via format specifiers but the new formatter specializations for chrono types are localized by default and don’t provide such control.

The problem

Example:

```
std::locale::global(std::locale("ru_RU"));

std::string s1 = std::format("{}", 4.2);
// s1 == "4.2" (not localized)

std::string s2 = std::format("{:L}", 4.2);
// s2 == "4,2" (localized)

using sec = std::chrono::duration<double>;
std::string s3 = std::format("{:%S}", sec(4.2));
// s3 == "04,200" (localized)
```

In addition to being inconsistent with the design of `std::format`, there is no way to avoid locale other than doing formatting of date and time components manually.

The problem

- Confusingly, some chrono format specifiers such as %S may give an impression that they are locale-independent by having a locale's alternative representation like %0S while in fact they are not.
- The implementation of [P1361] in [FMT] actually did the right thing and made most chrono specifiers locale-independent by default, for example:

```
using sec = std::chrono::duration<double>;
std::string s = fmt::format("{:%S}", sec(4.2));
// s == "04.200" (not localized)
```

- This implementation has been available and actively used in this form for 2+ years. The bug in the specification of chrono formatters in the standard and the mismatch with the actual implementation have only been discovered recently and reported in [LWG3547].

The solution

- Make chrono formatters locale-independent by default.
- Provide the L specifier to opt into localized formatting in the same way as it is done for all other standard formatters.

Before	After
`auto s = std::format("{:%S}", sec(4.2));` `// s == "04,200"`	`auto s = std::format("{:%S}", sec(4.2));` `// s == "04.200"`
`auto s = std::format("{:L%S}", sec(4.2));` `// throws format_error`	`auto s = std::format("{:L%S}", sec(4.2));` `// s == "04,200"`

Locale alternative forms

- Some specifiers (`%d %H %I %m %M %S %u %w %y %z`) produce digits which are not localized (`0123456789`) although `%S` is still using a localized decimal separator.
- They have an alternative form (`%Od %OH %OI %Om %OM %OS %Ou %Ow %Oy %Oz`) where the numerals can be localized, e.g. Japanese numerals 〇 一 二 三 四 五 ... can be used by a `ja_JP` locale.

- We do not propose to modify the specifiers, for example, `"{:L%p%I}"` and `"{:L%p%OI}"` should be valid specifiers producing 午後1 and 午後一 respectively.

The "C" locale

The "C" locale is used in the wording as a way to express locale-independent behavior. The C standard specifies the "C" locale behavior for `strftime` as follows.

> In the "C" locale, the E and O modifiers are ignored and the replacement strings for the following specifiers are:
>
> ```
> %a the first three characters of %A.
> %A one of Sunday, Monday, ... , Saturday.
> %b the first three characters of %B.
> %B one of January, February, ... , December.
> %c equivalent to %a %b %e %T %Y.
> %p one of AM or PM.
> %r equivalent to %I:%M:%S %p.
> %x equivalent to %y.
> %X equivalent to %T.
> %Z implementation-defined.
> ```

This makes it possible, as long as the `L` option is not specified, to format dates in environment without locale support (embedded platforms, `constexpr` if someone proposes it, etc).

SG16 polls

Poll: LWG3547 raises a valid design defect in [time.format] in C++20.

SF	F	N	A	SA
7	2	2	0	0

Consensus: Strong consensus that this is a design defect.

Poll: The proposed LWG3547 resolution as written should be applied to C++23.

SF	F	N	A	SA
0	4	2	4	1

Consensus: No consensus for the resolution

SA motivation: Migrating things embedded in a string literal is very difficult. There are options to deal with this in an additive way. Needless break in backwards with compatibility.

SG16 polls

SG16 recognized that this is a design defect but was concerned about this being a breaking change. However, the following facts were not known at the time of the discussion:

- The implementation of [P1361] in [FMT] is locale-independent. This was the only implementation available for 2+ years and was cited as the only source of implementation experience in the paper.

- Both `%S` and `%OS` depend on locale and there is no locale-independent equivalent.

- The chrono formatting in the Microsoft's implementation has only been merged into the main branch on 22 April and has bugs that will require breaking changes.

- Some chrono types are partially localized, e.g. `month_day_last{May}` may be formatted as `Mai/last` in a German locale with only month localized.

LEWG polls

Poll: Revise D2372 to keep the ostream operators for chrono formatting dependent on the stream locale

SF	F	N	A	SA
10	8	2	0	0

Outcome: Strong Consensus in Favor

Poll: LEWG approves of the direction of this work and encourages further work as directed above with the intent that D2372 (Fixing locale handling in chrono formatters) will land in C++23 and be applied retroactively to C++20

SF	F	N	A	SA
14	8	0	0	0

Outcome: Unanimous approval

Implementation experience

The L specifier has been implemented for durations in the fmt library ([FMT]). Additionally, some format specifiers like S have never used a locale by default so this was a novel behavior accidentally introduced in C++20:

```
std::locale::global(std::locale("ru_RU"));
using sec = std::chrono::duration<double>;

std::string s = fmt::format("{:%S}", sec(4.2));
// s == "04.200" (not localized)
```

The proposed fix including LEWG suggestion and 2 drive-by locale bug fixes has been implemented and submitted to the Microsoft standard library.

Impact on existing code

Changing the semantics of chrono formatters to be consistent with standard format specifiers (format.string.std) is a breaking change.

At the time of writing the Microsoft's implementation recenly merged the chrono formatting into the main branch and is known to be not fully conformant.

For example:

```
using sec = std::chrono::duration<double>;
std::string s = std::format("{:%S}", sec(4.2));
// s == "04" (incorrect)
```

Wording

All wording is relative to the C++ working draft [N4885].

Update the value of the feature-testing macro `__cpp_lib_format` to the date of adoption in [version.syn]:

Change in [time.format]:

chrono-format-spec:
fill-and-align$_{opt}$ *width*$_{opt}$ *precision*$_{opt}$ *L*$_{opt}$ *chrono-specs*$_{opt}$

2 Each conversion specifier *conversion-spec* is replaced by appropriate characters as described in Table [tab:time.format.spec]; the formats specified in ISO 8601:2004 shall be used where so described. Some of the conversion specifiers depend on ~~the locale that is passed to the formatting function if the latter takes one, or the global locale otherwise.~~ a locale. If the *L* option is used, the locale is the locale passed to the formatting function, or otherwise the global locale. If the *L* option is not used, the "C" locale is used. If the formatted object does not contain the information the conversion specifier refers to, an exception of type `format_error` is thrown.

Wording (drive-by)

6 If the *chrono-specs* is omitted, the chrono object is formatted as if by streaming it to `std::ostringstream os` with a locale imbued and copying `os.str()` through the output iterator of the context with additional padding and adjustments as specified by the format specifiers. If the *L* option is used, the locale is the locale passed to the formatting function, or otherwise the global locale. If the *L* option is not used, the "C" locale is used.

(Using the locale passed to the formatting function is a drive-by fix.)

Wording (operator<<)

Change in [time.clock.system.nonmembers]:

```
template<class charT, class traits, class Duration>
  basic_ostream<charT, traits>&
    operator<<(basic_ostream<charT, traits>& os,
               const sys_time<Duration>& tp);
```

...

2 *Effects*: Equivalent to:

```
  auto const dp = floor(tp);
  return os << format(
    os.getloc(),
    STATICALLY-WIDEN("{:%F %T}""{:L} {:L}"),
    year_month_day{dp}, hh_mm_ss{tp-dp});
```