Mutable Proposal: New Format Conversion Specifiers for Clocks

(Draft 02)

Document Number: <WHERE DO I GET THIS?>
Date: 2023-05-22
Audience: Library Evolution Working Group
Reply To: Simon Hill <protogrammer at gmx dot com>

Contents:

No AI was used in any way in the creation of any part of this document.

Introduction:

C++20's `std::[format]` family of functions take a format string and a set of value-arguments, replace each conversion specifier substring in the format string with a text representation of a value-argument (formatted according to a combination of the specific conversion specifier used and the type of the value-argument), then output this new string.

The set of conversion specifiers (and associated format behaviors) differs depending on the type (or type category) of the respective value-argument (eg between scalars and clocks).

This proposal introduces a number of potential additional conversion specifiers, and associated formats behaviors, that apply to `std::chrono` clock types.

The four behavior/specifier suggestions in this mutable proposal are:

A) Two-digit zero-padded truncated-integer decimal seconds: `%o`.
B) Variable-width truncated-integer decimal seconds-since-epoch: `%s`.
C) Decimal fractional component of seconds without radix/decimal point: `%f`.
D) Three-digit zero-padded truncated-integer decimal milli-, micro-, and nano-seconds: `%K`, `%k`, and `%L` respectively.

These proposed new format behaviors are, except for (C), frequently used elsewhere in computing, although the conversion specifier substrings are mostly unable to match common usage elsewhere because those strings are already assigned.

Each of these suggestions is independent of the others, and, if possible, each suggestion should be considered (or voted on) separately.

NOTE: The current set of `std::chrono` clock type conversion specifiers is listed in §29.12:6 of the C++ specification.

Rationale:

Using text strings to specify human-readable formats for time values is very common in computing, both in programming code and in end-user interfaces. Because of this, there are some common practices relating to what the substring format specifiers in these text strings mean, and how they are formatted.

The C++20 set of format specifiers for `std::chrono` clocks lacks some common format quantities, and has an unusual and, IMHO, inconsistent format for the *seconds* quantity.

I believe C++ would benefit from having these quantities, and having an additional *seconds* format (A) that matches common practice. Their common inclusion elsewhere speaks to their usefulness, and also suggests that developers will be familiar with them.

While it is possible to achieve formatting like this by *modifying the arguments* passed to the format function, this workaround is more verbose, less flexible, and, IMHO, less clear and tidy. Also, when external format strings are used, this workaround is not possible.

Examples (from personal experience and web search) where some or all of the suggested format behaviors/quantities are used:

Example	Two-digit zero-padded truncated-integer seconds	Seconds since epoch	Milli, micro, nano
XFCE clock	%S	%s	-
QT formatDateTime()	ss	-	zzz
.net formatting	ss	-	fff / ffffff
GLib / gnome clock extension	%S	%s	%f (6d micro)
GNU (eg ls)	%S	-	%N (9d nano)
National Instruments	<%S>	-	<%<digits>u>

I have not seen any example elsewhere of a non-truncated-integer seconds format.

For subsecond-precision clocks, C++ currently uses real-number seconds rather than truncated-integer seconds. To prevent subsecond digits being shown, you need to ensure the given argument type doesn't have subsecond precision (easily (but verbosely) done via a conversion such as `std::chrono::floor<std::chrono::seconds>()`).

Given the prevalence of `%S` for integer seconds, I believe the current behavior of C++ would be surprising to many developers.

Per-Suggestion Rationales:

(A). [Two-digit zero-padded truncated-integer decimal seconds]
This is the standard method of displaying seconds in most UI elements, even when using subsecond precision. In <format>, the other sub-day quantities (hours and minutes) are already two-digit truncated-integer so this improves consistency. When using a subsecond-precision clock value, this is less verbose than manually casting.

Letter `o` is derived from sec**o**nds (because `s` is commonly used for (B), and all better options are taken IMHO).

(B). [Variable-width truncated-integer decimal seconds-since-epoch]
This is common elsewhere, and is useful for easy and problem-free comparing and sorting of items (eg filenames) by time.

Letter `s` is chosen here because it is commonly used elsewhere for this purpose.

(C). [Decimal fractional component of seconds]
These can be used to extend the granularity of seconds-since-epoch, to allow for custom radix point use, and to help modularize implementation of the current %S behavior (which can be defined now as equivalent to `%o.%f`).

Letter `f` is derived from **f**raction.

(D). [Three-digit zero-padded truncated-integer decimal milli-, micro-, and nano- seconds]
These give programs easier control of precision and clarity in subsecond format, eg allowing for user-choice of digit separators. <chrono> already has explicit duration types for milli, micro, and nano. While even smaller metric prefixes could be used, I haven't found a precedent for them anywhere, and, as of 2023, there currently exist 10 smallizing metric prefixes (down to quecto) which would almost exhaust the available single-letter specifiers.

Letters `K`, `k`, and `L` are chosen because K and L are unused for both cases, and they are consecutive, and I can't find a better reason for any other letters.

Downsides

The downsides seem IMHO to be minor: A tiny increase in binary size, a negligible increase in runtime size equivalent to one or two more if statements per specifier, a tiny increase in C++ specification length, and a small amount of implementation work.

Conclusion

The main benefits of this proposal are providing common/familiar usages (A,B,D), the availability of these format behaviors/quantities when using non-local format strings (A,B,C,D), consistency (A), flexibility (C,D), and reduced verbosity (A).

Miscellaneous Details:

The main affected functions are std::{format, format_to, format_to_n}, and their respective std::vformat... versions.

For interest's sake, here is a Conversion Specifier Alphabetical Map for std::chrono clock types showing the proposed changes:

Letter	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Uppercase	u	u	u	u	p	u	u	u	u	-	D	D	u	-	p	-	u	u	u	u	u	u	u	u	u	u
Lowercase	u	u	u	u	u	C	u	u	-	u	D	-	u	u	A	u	u	u	B	u	u	-	u	u	u	u

- `-` = Unused (6)
- u = Currently used (38)
- p = Currently used as prefix (2)
- `A,B,C,D` = Proposed new use (6)

This proposal does use up a fair portion of the available character real-estate, but these behaviors are common elsewhere and IMHO it's unlikely that better candidate behaviors will be displaced by these.

[Decimal fractional component of seconds] (C) may be slightly harder to implement than the other suggestions, because there is no equivalent formatting behavior for scalar types that can be used. It may be worth considering also adding a new conversion specifier for the fractional component of *scalars* as this could be useful, eg for custom radix separator.

<format>'s parsing behavior must also be modified in the specification to match these changes.

Implementation Example:

%o is easily implemented in GCC by this diff:

[GCC PROJECT]/libstdc++-v3/include/bits/chrono_io.h

```
*** 382,387 ****
--- 382,388 ----
        __needed = _Duration;
        break;
      case 'S':
+     case 'o':
        __needed = _TimeOfDay;
        __allowed_mods = _Mod_O;
        break;
***************
*** 615,620 ****
--- 616,627 ----
      case 'S':
        __out = _M_S(__t, __print_sign(), __fc, __mod == 'O');
        break;
+     case 'o':
+       // %o  The second as a decimal number.
+       // %Oo Locale's alternative representation.
+       __out = _S_dd_zero_fill(_S_hms(__t).seconds().count(),
+               __print_sign(), __fc, __mod == 'O');
+       break;
      case 'u':
      case 'w':
        __out = _M_u_w(__t, std::move(__out), __fc, __c, __mod == 'O');
```

(the functionality is copied and adapted from '%M' (minute specifier))
(the case order is inconsistent, because the example originally used 's')

Impact On the Standard:

This proposal requires no changes to the C++ core language.
This proposal requires very minor changes to the "time.format" standard-library section of the C++ standard (§29.12:6).
This proposal requires no changes to other parts of the C++ standard library.
This proposal is easy to implement.

Proposed Specification Changes:

In [§29.12]

type : one of
```
+ a A b B c C d D e f F g G h H I j k K L m M n
+ o p q Q r R S s t T u U V w W x X y Y z Z %
```

In [§29.12:6: Table 102]: Meaning of conversion specifiers [tab:time.format.spec]

Add these rows in their appropriate alphabetical position:

%o	Seconds as a decimal integer. If the number of seconds is less than 10, the result is prefixed with 0. The modified command %Oo produces the locale's alternative representation.
%s	Seconds since the epoch as a decimal integer. The modified command %Os produces the locale's alternative representation.
%f	If the precision of the input can be exactly represented with seconds, then the format is an empty string. Otherwise, the format is the fractional part of a decimal floating-point number with a fixed format and a precision matching that of the precision of the input (or to a microseconds precision if the conversion to floating-point decimal seconds cannot be made within 18 fractional digits). The integer component of the value and the decimal point are not represented. The modified command %Of produces the locale's alternative representation.
%K	Milliseconds as a decimal integer. If the number of milliseconds is less than 10, the result is prefixed with 00. If the number of milliseconds is between 99 and 10 (inclusive), the result is prefixed with 0. The modified command %OK produces the locale's alternative representation.
%k	Microseconds modulus 1000 as a decimal integer. If the number of microseconds is less than 10, the result is prefixed with 00. If the number of microseconds is between 99 and 10 (inclusive), the result is prefixed with 0. The modified command %Ok produces the locale's alternative representation.
%L	Nanoseconds modulus 1000 as a decimal integer. If the number of nanoseconds is less than 10, the result is prefixed with 00. If the number of nanoseconds is between 99 and 10 (inclusive), the result is prefixed with 0. The modified command %OL produces the locale's alternative representation.

Note: Mostly just copied, pasted, and adapted from existing rows. The text could be simplified.

In [§29.12:6: Table 103]: Meaning of parse flags [tab:time.parse.spec]

Add these rows in their appropriate alphabetical position:

%o	The seconds as a decimal number. The modified command %*No* specifies the maximum number of characters to read. If *N* is not specified, the default is 2. Leading zeroes are permitted but not required. The modified command %Oo interprets the locale's alternative representation.
%s	Seconds since the epoch as a decimal number. The modified command %*Ns* specifies the maximum number of characters to read. If *N* is not specified, the default is 10. Leading zeroes are permitted but not required. The modified command %Os interprets the locale's alternative representation.
%f	The fractional part of seconds as a decimal number. The modified command %*Nf* specifies the maximum number of characters to read. If *N* is not specified, the default is 9. Leading zeroes are permitted but not required. The modified command %Of interprets the locale's alternative representation.
%K	The milliseconds as a decimal number. The modified command %*NK* specifies the maximum number of characters to read. If *N* is not specified, the default is 3. Leading zeroes are permitted but not required. The modified command %OK interprets the locale's alternative representation.
%k	The microseconds as a decimal number. The modified command %*Nk* specifies the maximum number of characters to read. If *N* is not specified, the default is 3. Leading zeroes are permitted but not required. The modified command %Ok interprets the locale's alternative representation.
%L	The nanoseconds as a decimal number. The modified command %*NL* specifies the maximum number of characters to read. If *N* is not specified, the default is 3. Leading zeroes are permitted but not required. The modified command %OL interprets the locale's alternative representation.

Note the altered *default characters to read*. `%f`=9 (nanoseconds in a second log 10). `%s`=10 (32-bit int max log 10). `%K` / `%k` / `%L`=3 (metric prefix step log 10).

Note: If only some of the suggestions in this proposal are accepted, only rows pertaining to accepted suggestions should be added to the specification. (this is probably obvious).

Acknowledgements:

Simon Hill.

References:

C++ Specification: §29.12:6.
Earlier Draft Proposal: https://lists.isocpp.org/std-proposals/att-6515/std-format-truncated-seconds-only.txt

Format Examples:

- **XFCE clock**: https://docs.xfce.org/xfce/xfce4-panel/clock
- **QT formatDateTime**: https://doc.qt.io/qt-6/qml-qtqml-qt.html#formatDateTime-method
- **.net formatting**: https://learn.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings
- **GLib**: https://github.com/lazka/pgi-docs/blob/master/GLib-2.0/classes/DateTime.html
- **GNU**: https://www.gnu.org/software/coreutils/manual/html_node/Time-conversion-specifiers.html
- **National Instruments**: https://www.ni.com/docs/en-US/bundle/labview/page/glang/codes_for_time_format_str.html