Document number: N2602=08-0112

Alisdair Meredith
Roger Orr
Anthony Williams
on behalf of BSI Panel 2008-02-21

BSI Requirements for a system-time library in C++0x

Requirements

The following is a list of requirements that BSI believe must be met by any library intended for use in system-time related tasks in C++0x, such as specifying timeouts for locks. The goal is to deliver the minimal set of requirements to specify a complete library that is fit-for-purpose. Features beyond those strictly necessary for system-time usage should be removed, and often we place a negative requirement that such features are specifically excluded from an acceptable library.

The formal list of requirements follows. A rationale section follows, explaining how we arrived at our choices.

1. The requirements for a *system time* form a distinct domain from calendrical and scientific/engineering use.
2. The use of templates and concepts in the documented API should be minimized.
3. C compatible ABI. If possible the new types and methods should be usable from C code.
4. The library shall encompass three concepts : clocks, instants/time-points, and durations
    a. clocks are a source of time instants
    b. instants denote a specific time local to the thread or process
    c. durations are a measure of the 'distance' between two instants
5. At least two kinds of clock shall be conditionally supported:
    a. Monotonic clocks are always incrementing
    b. 'real-time' clocks may be adjusted forwards/backwards by the environment
6. A further two categories of clock shall be conditionally supported:
    a. 'Global' clocks maintain a consistent epoch across threads within the same process
    b. 'Local' clocks need not maintain a consistent epoch between threads.

7. For each clock-type there shall be a mapping to a single instant type and single duration type appropriate for that clock. These two types shall be distinct from each other. [e.g. embedded `typedef`s or a traits class]

8. Clocks shall support an API to retrieve the 'current instant'

9. Unsynchronized reads of the current instant shall be valid.

10. Clocks define an epoch which is a fixed reference instant.

    a. 'Global clocks' share the same epoch across all threads in the same process.

    b. 'Local clocks' have no specified relationship between epoch in different threads.

11. Each clock shall define a minimum resolution for its duration and instant types. [Note: This is not the resolution of the clock, but of the instant/duration types.]

12. The minimum resolution can be no less coarse than a nanosecond.

13. The maximum duration must be at least 10 * 365 * 86400 seconds for a global clock, and 86400 seconds for a local clock.

14. Exceeding the max instant of a global clock is deemed exceeding implementation limits. A local clock shall wrap around.

15. The minimum and maximum value of instant must differ by the maximum expressible duration.

16. For two reads of the current system instant A and B from a 'global monotonic clock', if A happens before B then the value of A is not greater than the value of B. [1.10p8 N2588]

17. A real-time clock's value may be changed by a library API or external agents. For a 'global' clock, two reads of the current system instant A and B, if A happens before B then the value of A is not greater than the value of B [1.10p8 N2588], unless the clock is 'adjusted' between calls.

18. The library shall specify the behaviour in the absence of an intrinsic clock. [We make no claim on what that behaviour should be, merely insist the library explicitly acknowledge and address the problem]

19. There shall be no conversion [constructor or conversion operator], implicit or explicit, between fundamental types and time durations.
    [Alternative phrasing from Anthony:
    "Whenever you construct a duration, the scale/unit is always explicit"]

20. We shall be able to query and create duration objects in units of seconds, milliseconds, microseconds and nanoseconds. The values are returned as an integral type.

21. Resolutions of minutes and above fall into the calendrical domain and shall NOT be supported.

22. The internal representation to be opaque.

23. Arithmetic operations and comparisons on the duration and instant types are exact.

24. The following arithmetic operations shall be supported:

    a. instant + or - duration => instant

    b. duration +/- duration => duration

    c. duration multiply and divide by scalar => duration [note division may leak implementation details]

    d. instant - instant => duration

25. Comparisons (duration *op* duration) and (instant *op* instant) are supported for *op* in <, <=, >, >=, ==, !=

26. Durations and instants shall have a strict weak order, even if the library proposes special values analogous to 'infinite' or 'nan'.

27. numeric_limits shall be specialised for all duration and instant types.

    a. duration

        - min() returns the largest negative duration

        - max() returns the largest duration

    b. instant

        - min() returns the first instant

        - max() returns the last instant

Notes

The following issues were considered by BSI, and should also be considered by any successful library proposal. However, the BSI consensus was that so long as a reasonable answer was provided, we did not want to preclude any of the possibilities.

- We prefer chapter 20.7 rather than an extra chapter for the system-time library.
- Conversion functions to and from C-time is at the library's discretion
- Error reporting, handling of out of range values and overflow is not in scope of these requirements but shall be addressed by the library spec in some form

- An implementation may wish to state the likely precision (and accuracy) of instant values.
- Maximum resolution may be useful, even if exact value is not meaningful.
- It may be desirable to know the granularity of durations, however given the nature of a preemptive OS this information is likely to be of little use.
- We neither mandate nor oppose streaming of duration and instant objects.
- We are looking for something with a relatively cheap implementation cost (eg implementable as 64 bits)
- The resolution of instant/duration types and their clocks need not be the same (and typically won't be)
- wait_until(instant) - if clock changes you wait until the new time (unless clock is monotonic) wait_for(duration) - if clock changes you still get the original delay
- We also want wait_until(instant) on monotonic clock.

Rationale

Here we spell out the rationale and some of the ideas and implications of the requirements above. We separate this out so that the main list is easier to read and cross-reference in later papers.

1) The requirement is for a system time in a distinct domain.

> A *system-time* library has different requirements from both calendrical time and scientific/engineering use. Its design should not be constrained to be upwardly compatible with alien requirements from other domains.

2) The use of templates and concepts in the documented API should be minimized.

> There was strong feeling that using system time values should not require writing function templates. [Templates may be appropriate for other time types, such as calendrical or scientific ones.]

3) C Compatible ABI

> The threads proposals have generally tried to retain the possibility of C compatibility and the system time should do the same. We have an open mind about the link to C's existing time functions.

4) The library shall encompass … instants/time-points and durations.

> For a system time library there is a clear distinction between the two. Mandating different types allows for compile time detection of invalid usages.

4b.) Instants represent a specific time local to that process or thread.

We do not mandate that instants or durations can be transferrable between machines or processes. Nor that 'epoch' is consistent between two runs of the same process on the same hardware.

However some implementations might choose to implement a fixed epoch.

5) At least two kinds of clock may be conditionally supported, Monotonic and 'real-time'.

Monotonic clocks are usually easier and safer to use as arithmetic operations produce more consistent results. 'real-time' clocks provide some way of relating the system time to wall clock time for implementations where these are related.

However, monotonic clocks are not guaranteed to be available on all systems, so conditional support is the most we can require.

6) 'Global' clocks maintain a consistent epoch across threads within the same process, 'local' clocks need not maintain a consistent epoch between threads.

Local clocks provide sufficient functionality for timeouts, our primary use case, and there may be implementations where they are significantly easier to implement, or cheaper to use, than global clocks. In particular, 'local' monotonic clocks are more widely available than 'global' monotonic clocks: a local monotonic clock could be implemented using a per-CPU tick count, for example.

7) For each clock type there shall be a mapping to a single instant type and single duration type appropriate for that clock and distinct from each other.

A single instant type and a single duration type make for simple code without templates. This is important in drawing in the whole spectrum of C++ users, not just those comfortable with template heavy libraries like STL. Thread library primitives must be teachable to beginners learning the language, as well as serving the needs of the experts.

It does not make sense to use instant types from one clock type in calls for another clock type, so distinct types allow for overloading and prevent accidental misuse.

It is not unreasonable that different clocks share the same duration type, so the requirements state nothing about the type-mapping between different clocks – that remains the library designers' choice.

9) Unsynchronized reads of the current instant shall be valid.

Without this guarantee explicit synchronisation would be required for portability.

12) The minimum resolution can be no less coarse than a nanosecond.

We consider a resolution of less than a nanosecond would not be acceptable for modern systems. Some systems may wish to support resolutions higher than this, but defining a minimum makes it easier to write portable code.

We believe the library should specify this resolution explicitly, rather than leave it implementation defined. Portable code should be simple to write, not require querying various traits.

14) Exceeding the max instant of a global clock is deemed exceeding implementation limits. A local clock shall wrap around.

The primary usage of a local clock is expected to be for timeouts on waits, or similar operations. Provided the wrap-around period is sufficiently long, a monotonic clock with wrap-around is sufficient for this purpose, and can be provided on many systems that do not have a global monotonic clock. The `GetTickCount()` API on Windows is monotonic, but wraps around, and many Linux systems can provide a monotonic clock with wrap-around, even when they do not have a global monotonic clock.

15) The minimum and maximum value of instant must differ by the maximum expressible duration.

Given any two valid instants `t1` and `t2`, `t1-t2` must always evaluate to a valid duration, without overflow.

20) We want to be able to query and create duration objects in units of seconds, milliseconds, microseconds and nanoseconds. The values are returned as an integral type.

The initial intended use is for system time/timeouts rather than a calendar time. We expect the units to be specified in terms of naming, which makes it clear what usage and precision is intended.

When we discussed scales even as small as minutes there was disagreement on how or whether to handle leap seconds and other such issues. These are easily avoided in a system time library by simply not raising the question.

24) The following arithmetic operations shall be supported …

Certain operations – for example addition of instants -- are an error and this should be detected at compile time.

26) We want durations and instants to have a strict weak order.

This produces well defined comparisons and allows times and durations to be stored in sets, and use as keys, etc.