Why `duration` Should Be a Type in C++0X

Document #: WG21/N2526 = J16/08-0036
Date: 2008-01-30
Revises: None
Project: Programming Language C++
Reference: ISO/IEC IS 14882:2003(E)
Reply to: Walter E. Brown<wb@fnal.gov>
Marc Paterno <paterno@fnal.gov>
Computing Division
Fermi National Accelerator Laboratory
Batavia, IL 60510-0500

Contents

Numbers are the product of counting. Quantities are the product of measurement.

— GREGORY BATESON

1 Introduction

1.1 Motivation

This paper is motivated by recent drafts of proposals for the threads portion of the C++0X standard library. In particular, interfaces to certain threads control functions (*e.g.*, `sleep()`) are designed so as to make use of a *duration* metric. Because *duration* is classified in the SI Brochure[1] as one

[1]The International Bureau of Weights and Measures (BIPM), established by international treaty in 1875, has as its mission "to ensure worldwide unification of measurements." Among its other tasks, the BIPM "establish[es] fundamental standards [. . .] for the measurement of the principal physical quantities." Its publication, the Système International d'Unités, the SI (known in English as the International System of Units), is the worldwide standard for all matters related to "current best measurement practice." The complete text of the reference document, "commonly called the SI Brochure," can be freely downloaded from http://www.bipm.org/en/si/si_brochure/general.html. It, in turn, is based on the International System of Quantities, ISQ, published in *Quantities and Units*, International Standard ISO 31:1992(E), ISBN 92-67-10185-4.

of the seven "base quantities" of the SI, we believe it is important that, as a nascent International Standard, C++0X be consistent with other International Standards, and in particular with the SI's long-established codified practices regarding durations and similar quantities. Unfortunately, the current threads proposal seems inconsistent with such practices, and so we are very concerned about the direction taken.

We previously made our concerns known via email to the Editorial Committee appointed as part of Kona's LWG Motion 14. This was done as part of our commitment (pursuant to that same Motion) to review the work of the Editorial Committee. However, that Committee concluded that it was not within their charge to consider the issues thus raised and so took no action regarding these concerns. After reviewing the Editorial Committee's second draft, we wrote on 2008-01-23:

> It seems that the Editorial Committee has made no adjustments to the text of the draft in response to the comments and the subsequent clarifications that we had offered regarding the first draft of the paper. Because we consider the underlying issues to be so serious as to constitute a fatal defect in the proposal, and because [. . .] time grows short, we respectfully recommend that the paper be not forwarded to the Project Editor in its current form and that it instead be remanded to the full Committee for further discussion and direction.
>
> We believe that the Editorial Committee does have the authority to take the above recommended actions, and further believe that such actions would be both responsible and not inconsistent with the charge to the Editorial Committee. We look forward to a full and fair airing of the issues, so that the threads library proposal can be perfected and then become part of C++0X.

A second member of the Review Committee wrote later that day to share in our concerns. However, the Editorial Committee chair responded on the same date that he was "not sufficiently concerned about the risk that results from putting this [draft] into the WP that [. . .] it outweighs the benefit of having a threads API in the working paper so that we have more time to think about and fix secondary threads issues."

1.2 Overview

Because we strongly disagree with the above outcome, we are formalizing and expanding our concerns in this paper. Section 2 will identify what we believe to be the important requirements that must be satisfied by any library that provides or makes use of durations and related quantities. Section 3 will give a brief overview of the proposed date-time facility within the threads library proposal, while Section 4 will present a discussion and critique of that facility's design. Section 5 will describe a few alternative designs that we consider preferable. We summarize in our concluding Section 6. For reference purposes, Appendix A presents some basic information about the SI, and defines the nomenclature used throughout the body of this paper.

2 Desiderata

2.1 Additive properties

In both commercial and scientific applications, values of quantities can come about as the result of an observation (a *measurement*), or as the result of a calculation involving other quantity values. Sums and differences of such quantities are meaningful if and only if the operands' quantities are of the same kind (*i.e.*, are commensurate, have the same dimension).

We consider this commensuration property to be an important requirement. For example, the sum or difference of two *length* quantities must always produce a length (or a commensurate equivalent such as a *perimeter*, a *height*, an *altitude*, or a *distance*). In contrast, two quantities

that are not commensurate can produce no meaningful sum or difference; any expression calling for same should be diagnosed as ill-formed.

2.2 Multiplicative properties

Products and ratios (quotients) are also important and sensible results of operations on quantity values. Unlike additive operators, these operations do not require that their operands' quantities be commensurate. For example, a length quantity can be divided by a duration quantity, and yields a *speed* quantity. Similarly, the reciprocal of a duration (*e.g.*, ticks per second) is a *frequency* quantity, and the ratio of two durations is a unitless quantity.

Indeed, the kind of the resulting quantity for multiplicative operations is in each case determined by the kinds of the operand quantities. As additional examples, the product of two length quantities yields an *area* quantity, the product of three lengths (or of a length and an area) yields a *volume*, and an *acceleration* is the quotient of a length and the square of a duration. Such behaviors are more easily represented by inspecting the dimensions of each quantity: an area's dimension is L^2, a volume's dimension is L^3, an acceleration's dimension is L/T^2, and a frequency's dimension is $1/\mathrm{T}$.

Finally, as an important specific case of the product of two quantities, recall that the SI speaks of a quantity's value "as the product of a number and a unit." As a number is considered a dimensionless quantity, and a unit is by definition a quantity, this is a sensible product, and one that we believe important to preserve.

2.3 Constancy of units

The SI establishes that a quantity's value is always expressed in terms of some unit that is used as a reference. The SI further demands (although it can be easily deduced) that all such units need themselves be values of the same quantity as that for which it serves as a reference.

Historically, among the major problems that lead to the formation of the BIPM was that units by the same name, but in different jurisdictions, often denoted distinct quantity values.[2] Indeed, one of the SI's greatest contributions to moderm mensuration and related technology has been the standardization of what has been termed a "scientific system of units" (in contrast to older so-called "customary units").

Therefore, it has become axiomatic that each unit shall denote a constant quantity. We believe this to be an important and useful property.

2.4 Library coherence and interoperability

We firmly believe that any library providing quantity services should interoperate with and be conveniently usable by all libraries requiring such services. This opinion seems shared by the LWG, as it was the motivation for incorporating a date-time facility to serve the needs of the threads library. Indeed, it is our understanding that other libraries will be proposed (*e.g.*, for TR2) that will make use of duration or similar services.

[2] "In antiquity, systems of measurement were defined locally, the different units were defined independently according to the length of a king's thumb or the size of his foot . . . " [Wikipedia, *Systems of measurement*, 2007-01-08]. Even today, we have units that are in common use, yet that are overloaded with multiple definitions: *gallon*, for example, could denote the U.S. liquid gallon, the U.S. dry gallon, or the Imperial (UK) gallon.

It is for this reason that we believe it important to consider more general use cases beyond the needs of the threading library.

3 Overview of proposed date-time facility

The current threads proposal incorporates N2328: "Proposal for Date-Time Types in C++0x To Support Threading APIs," a minimalist version of the date-time library that had been proposed in N1900: "Proposal to Add Date-Time to the C++ Standard Library 0.75" and N2058: "Proposed Text for Proposal to add Date-Time to the Standard Library 1.0." All these proposals are based on the Boost Date-Time library, self-described as "an open source C++ library developed by CrystalClear Software."

This date-time part of the threads proposal defines the following terms:

> A *time point* represents a dimensionless [*sic*!] instant in the time continuum. A *time duration* represents a length of time unattached to any time point. Time points and time durations have a *resolution* which is their smallest representable time duration. Time points have an *epoch* or start of a given time scale.

Duration requirements are, in turn, principally defined via a requirements table (*i.e.*, by a concept), and can be realized by any type that meets those requirements (*i.e.*, by any type that models that concept). In particular, the threads proposal includes text to standardize six such types: `std::nanoseconds`, `std::microseconds`, `std::milliseconds`, `std::seconds`, `std::minutes`, and `std::hours`. In addition, there are function templates to provide the usual comparisons between values of two such types, sums and differences of such values, products of one such value with a `long` value, and the quotient of a duration value with a `long` value.

There are no general requirements specified on types that correspond to a time point, but one such type (`system_time`) is mandated by the proposal.

4 Discussion and critique of proposed date-time facility

4.1 Unnecessary proliferation of templates due to units as types

Most importantly, the facility's design makes it inconvenient for both authors and users of this library facility and of any library seeking to extend this facility by representing, via concepts rather than via types, quantities of other dimensionalities. Among our greatest concerns is that the underlying design makes it unnecessarily difficult to deal with units. The multiplicity of types strongly suggests that any function taking a duration parameter must be expressed as a template in order to accept arguments that may be expressed via any of the duration types.

Any function of, say, n parameters, each of a quantity type, would require n template parameters, even if all the types were commensurate. It is the equivalent of wishing to write a function to calculate the perimeter of a triangle whose sides a, b, c may have types A, B, and C, where each may be a distinct unit (*e.g.*, meters, millimeters, and centimeters); this conceptual function would need to be implemented as a template in order to deal with the distinct units. It seems to us that a more convenient approach, for both authors and users, would be to have a single `length` type, to have each unit be a quantity of that type, and to express each of the triangle's sides as appropriate multiples of such a unit. Then a single function taking three `length` arguments and returning a `length` would suffice.

For the record, several generations of software in (at least) the physics community have successfully treated units as constant values, and have expressed quantities as products that involve these units. Geant4, "a toolkit for the simulation of the passage of particles through matter," is one current example of such software; see `http://www.geant4.org` for details.

4.2 Combinatorial explosion of types due to units as types

Another consequence of the current approach is, upon extension, the combinatorial explosion of types. To write a function calculating, say, a speed as the quotient of any of n lengths and any of m durations, would lead to the possibility of $n \times m$ result types. We thus believe that this approach fails to scale when used in multiplicative conjunction with other quantity types. Since SI specifies seven base quantities, and since common use will routinely involve several quantities, each raised to some small exponent (*e.g.*, energy is mass times length squared divided by duration squared), the number of types needed to represent quantities in even simple programs can quickly become huge.

4.3 Lack of user control over precision versus range

As described above, in this library facility's design, units are realized as types that model its duration concept. There is thus a multiplicity of types that are commensurate; the commensuration of values of these types is achieved via template technology. This design requires a choice between preserving range or preserving precision: For example, if we declare `ns` to be of type `std::nanoseconds` and `hr` to be of type `std::hours`, what type should result from their sum `hr + ns`? The proposal opts for the type that has "the finest [*sic*] duration," but we believe that there are valid use cases for either decision.

For most measurement purposes, it seems to us preferable to retain precision rather than dynamic range. In the absence of sufficient precision, a very large quantity added to a very small quantity should, we believe, usually yield the very large quantity. A more suitable library design would leave the choice of range versus precision to the client. It seems to us that it is the lack of such user control that has lead to the design decision to prefer precision over range.

4.4 Unintuitive results due to choice of integral representations

We are also concerned that `minutes(2.5) == minutes(2)` and that `minutes(2.5) != seconds(150)`. There seems, in the current specification, neither useful semantics nor diagnostics in the presence of floating point values.

Further, the defaults seem misleading in practice. The expression `3 * minutes()` appears to denote a three-minute duration, yet evaluates to the equivalent of `minutes(0)`, a zero-minute duration.

Incidentally (apropos of these examples), we note that the only requirement regarding the construction of a duration type is that the type be DefaultConstructible and CopyConstructible, with no specified semantics. It is clear from earlier documents that such expressions as `hours(1)` are generally intended to be valid and meaningful, and the individual specifications of the mandated types do provide this capability, yet the general duration requirements seem silent regarding this.

4.5 Superfluous distinction between time point and duration

Paraphrasing the definitions quoted above, this library facility treats a time point as a *position* in time relative to a designated *origin*, and a time duration as a *displacement* from one time point to another time point. We note that the facility provides types that correspond to one or to the other of these notions, thus treating *time point* and *time duration* as substantively different from each other. We disagree that these notions should be viewed in such dramatically differently manners.

It would certainly make sense to differentiate between a *displacement* and a *position* (whether speaking of time, space, or temperature) if different positions can denote displacements relative to different origins. This distinction is necessarily lost, and therefore moot, when there is only a single type that represents all positions. In other words, since this library facility standardizes

only a single type (`system_time`) to correspond to a time point, and thus provides but a single time origin (namely, the traditional UNIX epoch), we believe there is no need for distinct types to distinguish either:

- between distinct origins, or
- between a time point and a duration (distance in time).

While the need for such distinctions may be revisited in the context of TR2, we believe it is today unnecessary (especially given our current time pressures), and hence that it is premature to standardize a level of generality for which no need has been demonstrated in C++0X.

5 Alternatives

As we had previously written to the Editorial Committee, "We have thought of at least three alternatives, each of which we like better (for purposes of today's threads library) than what is now proposed." Parts of these alternatives are inspired by POSIX precedents that would have us use units implied by functions' names (*e.g.*, `sleep()` [implying seconds] and `nanosleep()` [implying nanoseconds]).

1. If there is a type `std::threads::duration`, then we can have and use constants of that type.

   ```
   namespace std::threads {
     constexpr duration  minute = ...;
     constexpr duration  second = ...;
   }
   ```

   And if the `sleep()` function can take a duration as its argument (which seems to be the intent), then [. . .] users can write:

   ```
   std::this_thread::sleep( 3 * std::threads::minute
                          + 30 * std::threads::second );
   std::this_thread::sleep( 3.5 * std::threads::minute );
   ```

   We emphasize that this needs only a single duration type, plus a few convenience constants of that type together with appropriate operators involving that type.

2. We can use an integral type instead (*e.g.*, `int64_t`), noting (perhaps via a naming convention or just via documentation) that all values are in, say, nanoseconds.

   ```
   namespace std::threads {
     constexpr int64_t  second = 1_000_000_000; // using Lawrence's notation
     constexpr int64_t  minute = 60 * second;
   }
   ```

   And if the `sleep()` function takes an `int64_t` as its argument, then users can still write:

   ```
   std::this_thread::sleep( 3 * std::threads::minute
                          + 30 * std::threads::second );
   ```

3. We can use an integral type (again, *e.g.*, `int64_t`), and (instead of any named constants) rename the function to indicate the implied units:

   And if each `sleep()` function takes an `int64_t` as its argument, then users can write:

```
std::this_thread::nanosleep( 30_000_000_000 );
std::this_thread::microsleep( 30_000_000 );
std::this_thread::millisleep( 30_000 );
std::this_thread::sleep( 30 );  // in seconds
```

All the above are equivalent in intent.

6 Summary and conclusion

The Library Working Group has taken the position that there should be a single facility to provide a general date-time duration, and that this facility should serve the duration needs of the threads library and of other libraries. We agree with this direction.

We disagree, however, that the present design is adequate to serve the needs of such other libraries. In particular, we have shown that the "little bit" of the date-time library used by the threads library exposes a design that precludes using this date-time duration notion from being reusable in a units library. Further, without breaking future backward compatibility, the design seems to usurp important names and thus precludes any extension or replacement in a more suitable direction.

In particular, we believe in, and recommend adherence to, the following basic principles:

1. *Quantities* (such as *duration*) are candidates for abstraction as C++ types.
2. *Units* (such as *seconds*) are treated as constants of their respective *quantity* types.

We respectfully urge that the underlying issues be considered on a time scale consistent with C++0X.

7 Acknowledgments

We thank the Fermi National Accelerator Laboratory's Computing Division, sponsor of our participation in the C++ standards effort, for its past and continuing support of our efforts to improve C++ for all our user communities.

* * * * *

A Appendix: Nomenclature and basics of the SI

Note: Because the following definitions are compiled from several sources (as indicated), we have in some cases adjusted the typography (*e.g.*, by italizicing) for consistency and clarity.

A.1 QUANTITY, VALUE OF A QUANTITY

The *International Vocabulary of Basic and General Terms in Metrology* (VIM) defines a *quantity* as a "property of a phenomenon, body, or substance, to which a magnitude can be assigned."

The SI Brochure states in its opening paragraph, "The *value of a quantity* is generally expressed as the product of a number and a unit. The unit is simply a particular example of the quantity concerned which is used as a reference, and the number is the ratio of the value of the quantity to the unit."

A.2 BASE QUANTITY, DERIVED QUANTITY, DIMENSION

VIM defines a *base quantity* as a "quantity, chosen by convention, used in a system of quantities to define other quantities." It further defines a *derived quantity* as a "quantity, in a system of quantities, defined as a function of base quantities."

The SI uses seven base quantities. "Each of the seven base quantities used in the SI is regarded as having its own *dimension* . . . All other quantities are derived quantities, which may be written in terms of the base quantities by the equations of physics. The dimensions of the derived quantities are written as products of powers of the dimensions of the base quantities . . . "

For example, *length* and *duration* (also known as *time*) are two of the SI's base quantities, and *speed* is a derived quantity. Using L and T, respectively, as the symbols for the length and duration dimensions, the dimension for speed is given by L^1T^{-1}, typically algebraically simplified to LT^{-1} or L/T. (Technically, all seven dimensions should be represented, but we have elided for simplicity the remaining five whose dimensional exponents are zero.)

A.3 BASE UNIT, DERIVED UNIT

Corresponding to its seven base quantities, the SI defines seven *base units*. For example, the base unit corresponding to *length* is the *metre* (m), and the base unit corresponding to *duration* is the *second* (s). Further, "*Derived units* are products of powers of base units." Thus, the derived unit corresponding to speed is m^1s^{-1} (equivalently, m/s or metre per second).

A.4 DIMENSIONLESS QUANTITY, UNITLESS QUANTITY

As a special case, the SI refers to a quantity as *dimensionless* if each of its dimensional exponents is zero. Such quantities are considered to have "the unit one, 1"; however, this unit "is generally omitted in specifying the values of dimensionless quantities" and so these quantities are often informally described as *unitless*. Examples of such quantities include those that "have the nature of a count."

A.5 COMMENSURABLE QUANTITIES, QUANTITIES OF THE SAME KIND

Finally, in accepted scientific parlance, two quantities are *commensurable* (or *commensurate*) if values of one can be expressed in the same units as values of the other, *i.e.*, if they are measurable by a common standard. Equivalently, VIM defines the phrase *quantities of the same kind*, requiring that "Quantities of the same kind within a given system of quantities have the same dimension."