We shouldn’t rush to require either CHAR_BIT==8 or (u)intNN_t

Document number: P3635R0
Date: 2025-02-13
Project: Evolution Working Group, Library Evolution Working Group
Reply-to: Nevin “☺” Liber, nliber@anl.gov

Introduction

This is a rebuttal to P3477R3 There are exactly 8 bits in a byte.

Motivation and Scope

I have grave concerns on adopting P3477R3 for C++26.

The motivation is weak and contradictory, and every time it gets discussed we discover new information. In the past WG21 has pushed through things like deprecating volatile, and then embarrassingly had to roll back those changes in a later standard because the research behind it was incomplete. The first time P3477 was seen was in the pre-Wrocław mailing (the last possible time to be considered for C++26), and that isn’t enough time to ruminate on it given the speed at which new information is discovered.

The first time LEWG saw this was post-Wrocław in a mailing list review. Believing it to be a “slam dunk”, that review immediately asked for a forwarding poll. Yet new information was discovered: namely, that it doesn’t follow from CHAR_BIT==8 that (u)int16_t, (u)int32_t and (u)int64_t become required types. CHAR_BIT==8 doesn’t lead to all the other “power of 2” types existing. It is misleading to insist that P3477 is ONLY about CHAR_BIT==8. There is no evidence that either EWG or SG22 had discovered such information in their first sessions.

During the EWG discussion this past Monday in Hagenberg, new information was discovered that there exist platforms where (u)intNN_t is an extended integer type. While it was also claimed in that EWG session that P3477 requires no compiler or library changes anywhere (so why rush?), that assumes there is no library impact. But even with only 3 days to consider it, that doesn’t seem to be true.

Library has virtual no mechanisms to deal with extended integer types. For instance, there is not even a type trait to detect it! While one could write a fragile one (because its correctness would be subject to the whims of WG21) ,this directly contradicts the P3477 motivation that it simplifies coding.

Even ignoring the lack of a trait, a cursory look at our standard library shows there are other issues to consider.

For instance, we have overloads for abs (and similar but not identical ones for div):

constexpr int abs(int j); // freestanding
constexpr long int abs(long int j); // freestanding
constexpr long long int abs(long long int j); // freestanding

and sometimes

constexpr intmax_t abs(intmax_t); // if and only if intmax_t is an extended integer type

If we are adding a required type that may be an extended integer type, should we add an overload for it? Should that be in freestanding (is the abs(intmax_t) an oversight or deliberately not in freestanding?)? If it is a required type, implementors have no out, so we have to answer these questions as part of considering this proposal.

And if we make that addition, obviously this is not a zero impact proposal for library vendors, despite claims made in EWG. And because EWG had polls based on that claim, we would once again have to send it back to EWG.

More generally, what does a maximally portable user library which wants to do such overloading on required types have to do? Invent a trait and add a requires clause? Does such an overload also have to be concerned that intmax_t and intNN_t might be the same extended integer type? How does any of that simplify things??

Is this the only change we need to consider? Beats me. P3477 doesn't explore any of this. I certainly don't have the time this week to do more exploring, and even if I did, it isn't my proposal! While the line is fuzzy, generally LEWG doesn't do design on the fly either.

The latest revision, P3477R3, didn't even explore points that came up during the mailing list review.

For instance, it was brought up that the motivation that networking needs CHAR_BIT==8 everywhere could easily be solved by only requiring it for that upcoming library. And that library isn't even being standardized for C++26!

And every minute we spend on this proposal is a minute we cannot spend on considering things that could improve things for C++26 developers, especially for LEWG as our schedule is tight and already has many on time papers we are extremely unlikely to see.

Going over the motivation point by point:

- How can one write a truly portable serialization / deserialization library if the number of bits per bytes aren't known?

How can one write a truly portable serialization / deserialization library if one or more of the types it uses may or may not be an extended integer type? Does an implementation of this serialization library exist so we can play around with it?

- Networking mandates octets, specifying anything networking related (as the committee is currently doing with [P3482R0] and [P3185R0]) without bytes being 8 bits is difficult.

static_assert(CHAR_BIT==8) (or the equivalent in standardese) solves this, doesn't it?

- The Unicode working group has spent significant time discussing UTF-8 on non-8-bit-bytes architectures, without satisfying results.

Like all claims about performance, we need benchmarks, not anecdotes, to substantiate them.

- How do fread and fwrite work? For example, how does one handle a file which starts with FF D8 FF E0when bytes aren't 8 bits?

That seems like a question for WG14.

- Modern cryptographic algorithm and the libraries implementing them assume bytes are 8 bits, meaning that cryptography is difficult to support on other machines. The same applies to modern compression.

If the claim is that such platforms aren't targeted by modern C++ compilers, then it doesn't matter. Otherwise, like all programming problems, if they need the functionality they need to solve it.

My recommendation is that we reject this for C++26. As the Strongly Against stated in EWG in Hagenberg on Monday, it is a thorough waste of time.