


Towards Text in the Standard Library

Date: 2019-09-26
Project: Programming Language C++
Audience: Organization Sponsor
Reply-to: JeanHeyd Meneide<phdofthehouse@gmail.com>
Columbia University

This foundational work is needed in C++23 to enable non-experts to write ‘hello ’.

— Tom Honermann,
Study Group 16 Chair

Currently, C++ is bereft of the ability to go from legacy platform encodings to Unicode or even transform text between two encodings. This greatly inhibits C++ programs in communicating and inter-operating with external system resources. WG21’s Direction Group in P0939 named Unicode and Text Processing one of its priorities for the coming years [1], for which the Committee created Study Group 16 (SG16). SG16 is the Unicode and Text Processing arm of C++ with the goal of having a coherent, seamless experience with respect to encoding, decoding, transcoding and producing useful Unicode algorithms. After advances in updating the Unicode Standard [2] and `char8_t` [3] for C++20, Study Group 16 is ready to tackle larger problems, particularly in the library space.

Proposal P1629 Standard Text Encoding [4] was written with the goal of tackling one such large problem. The paper is a synthesis examining implementation experience from several high quality, open source library implementations (Mozilla Firefox, International Components for Unicode (ICU) [5], Boost.Text [6], libogonek [7], etc.) and implementers who have put much feedback into the system (Markus Scherer, Henri Sivonen, etc.).

With all of this work and feedback [8], SG16 now attempts to turn ideas, specification, and guidance into viable implementation. This proposal requests funding to work on an implementation of text to be available for early 2020 and to work towards inclusion into C++ for the 2023 cycle.

1 Background

I am a student at Columbia University who spends their free time working on high-quality C++ libraries [9] and participating in C++ Standardization [10]. After engaging in a Library Working Group "Issues Processing" session during the 2018 Rapperswil-Jona C++ Standards Meeting at the HSR Hochschule für Technik Rapperswil, I realized that I could help contribute to the Standard Libraries that make C++ both fast and easy to use.

After a successful Summer of Code 2019 with the Free Software Foundation working on Bit Abstractions for libstdc++ [11], I realized that the only way to reach Study Group 16’s professed goal of making text encoding available in time for C++23 was to furnish a high-quality implementation of what would go into a C++23 Standard Library.

I am an expert in API design for abstracting complex systems into easy to use libraries that:

- do not forsake the performance characteristics of the low-level and complex solutions to "leave no lower level language below C++";
- and, do not forsake the "Onion Principle" by providing incrementally easier layers that can be peeled off to get at the lower level details to avoid frustrating expert users.

SG16's Direction Paper [12] has rated transcoding and normalization as the #1 priority for forward progress in §5.1. Tom Honermann has specifically stated that P1629 and related work is a "high priority and target for early in C++23", with the proposal steadily building consensus in SG16 since the June 12th, 2019 SG16 Telecon Meeting [13]. Polls during SG16's inaugural meeting at the 2018 San Diego C++ Meeting ranked transcoding utilities at #1, with a repeat of such a priority in the 2019 Kona C++ Meeting. The work being done here is important and foundational to success with Text in C++.

My résumé is available upon request.

2 Implementation Deliverables

I don't write any software which runs only in English. I'm tired of writing the same code different ways all the time just to display a handful of strings.

— *Respondent,*
Herb Sutter's Top 5 Proposals Survey

The ultimate deliverable of this proposal is production of a high-quality implementation of low-level text handling, with powerful and simple abstractions that can be used to further work being done towards Unicode algorithms in Standard C++:

- **CORRECTNESS.** We are incredibly fortunate that there is a large Unicode Conformance test suite. Well-defined behavior is paramount and text is no exception to this rule.
- **EXTENSIBILITY.** User extension is key: C++ standard library implementers do not have the time to implement every single necessary encoding. They also do not know what internal or special encodings are used by individual companies, nor can the standard directly address their needs. Everyone should have a chance to supply additional encodings suitable for their workload.
- **PERFORMANCE.** If an implementation does not deliver acceptable speed by default, then it is not correct.

Unicode Algorithms need to work over code points, or scalar values (non-surrogate code points). Without the ability to for (safe) encoding and decoding from the world's various (legacy) encodings into Unicode, we lose the ability to transfer much of today's legacy text into Unicode. Lack of easily available utilities has been the biggest barrier to proper text handling in C++, and a standard solution is sorely needed.

2.1 Goals

The high-level goals are as follows:

- Implementing encoding, decoding and transcoding iterators and views for walking (immutable) storage of encoded text to produce Unicode Code Points.
- Implementing encoding, decoding and transcoding free functions for fast conversion from one text buffer to another.
- Provide normalization forms C and D of both canonical and compatibility flavors as specified in UAX #15 [14].

These goals were chosen due to their foundational independence from further work in Unicode. In particular, the three listed goals do *not* need to *tackle locale and internationalization challenges in any way, shape or form*. This makes it a strong candidate for potential inclusion in C++23 and lets separate, parallel work continue on locales.

The concrete goals are as follows:

- Have a production-ready, working implementation once all the goals and a number of stretch goals (particularly, SIMD optimization) are finished.
- Work across all 3 Standard Library implementations, including Visual C++, GNU Compiler Collection, and LLVM Clang.
- Licensed using the Apache v2.0 with LLVM Exception or Boost License *when full funding is reached*.
- Enable direct support for individuals or companies that provide additional funding, should they need extra help beyond what is provided by the open source implementation and support this proposal produces.

2.2 Time Frame

If sufficient funding is received, the timeline for this proposal is 6-12 months. The targeted working time is 40 hours a week under full funding. This would produce 760 - 1400 hours of work. The use of such time is as follows:

1. Implement basic `utf8`, `utf16`, `utf32`, `ascii` encoding objects: 40 hours.
2. Implement basic `wide_locale_execution` and `narrow_locale_execution` encoding objects: 40 hours. (NOTE: uses C standard functions for underlying implementation.)
3. Implement `encode_view`, `decode_view`, `transcode_view`, and corresponding iterator/sentinel pairs: 80 hours.
4. Implement `nf(k)c` normalization form, with iterators: 80 hours.
5. Implement `nf(k)d` normalization form, with iterators: 80 hours.
6. Implement `std::text::text_view<...>` and `std::text::text<...>` analogous types: 80 hours.

7. (STRETCH) Implement Shift-JIS (ISO 2022-JP) as a library example of a stateful encoding: 80 hours.
8. (STRETCH) Implement GB18030 as a library example of a Unicode Transformation Format that prioritizes compatibility with CJK (GBK) encodings: 80 hours.
9. Implement `encoding_scheme` objects for handling byte-oriented (network, file, etc.) streams: 35 hours.
10. Implement user extension ADL hooks for `decode`, `encode`, `transcode` and `transcode_one`: 35 hours.
11. Implement advanced `wide_locale_execution` and `narrow_locale_execution` encoding objects: 160 hours. (NOTE: Includes prying open relevant glibc and VC++ implementations of locale to properly transcode without loss of information from standards-compliant code on those platforms, with copious application of conditional compilation to ensure it works on all supported platforms.)
12. Benchmarking: implement use cases of ICU [5], `encoding_rs` [15], and `iconv` [16] for performance comparisons: 80 hours.
13. (STRETCH) Implementing the Strong Exception Guarantee for `std::text::text<...>` type: 120 hours.
14. (STRETCH) Fine-tuning using vectorization operations for transcoding pairs (UTF8 ↔ UTF16/32, ASCII → UTF8): 120 hours, any excess funding.
15. (STRETCH) WHATWG encodings [17]: any additional funding.
16. (STRETCH) Implement C free functions for transcoding between multi-byte and wide character literals and UTF8/16/32: any excess funding. (NOTE: Includes attending WG14 meetings to standardize such an interface, including sized and null-terminated conversion functions into glibc, MUSL, and VC++.)
17. (STRETCH) Upstream final implementation through `libstdc++`, `libc++`, and `VC++`. (NOTE: Includes attempting to fit into formatting guidelines, license adjustments, and more for each individual standard library.)

Every milestone includes providing tests for the implementation, and progress on the original P1629 proposal. Times shown are rough estimates based on current implementation knowledge as well as examination of the cited implementations and documents. Desired implementation strategy is shown in approximate order of priority. Unless explicitly directed otherwise, funds will be put into each item on the list from the first (top) to the last (bottom) priority. STRETCH goals are for if additional funding is available at that point in implementation progress, or if explicitly requested by a sponsor.

Check ins with sponsors will be sent out quarterly – sometimes sooner – depending on the goals / milestones being tackled during that time.

3 In Conclusion

I have the expertise, knowledge, and skills to deliver on the promise that text in C++ can be a powerful, fluid abstraction upon which many more expressive and wonderful algorithms may blossom.

I humbly request your consideration for funding to accelerate the development of Study Group 16's mission and for the good of Standard C++. If you have any questions, please do not hesitate in reaching out to me.

References

- [1] Howard Hinnant, Roger Orr, Bjarne Stroustrup, Daveed Vandevoorde, and Michael Wong. Direction for iso c++. <https://wg21.link/p0939>, 2019.
- [2] Steve Downey, Robot Martinho Fernandes, and JeanHeyd Meneide. Update the Reference to the Unicode Standard. <https://wg21.link/p1025>, 2018.
- [3] Tom Honermann. char8_t. <https://wg21.link/p0482>, 2019.
- [4] JeanHeyd Meneide. Standard Text Encoding. https://theohd.github.io/vendor/future_cxx/papers/d1629.html, 2019.
- [5] Inc. Unicode. Internationalization components for unicode. <http://site.icu-project.org/>, 2019.
- [6] Zach Laine. Boost.text. <https://tzlaine.github.io/text/doc/html/index.html>, 2018.
- [7] Robot Martinho Fernandes. libogonek: A c++11 library for unicode. <https://github.com/libogonek/ogonek>, 2016.
- [8] JeanHeyd Meneide and Henri Sivonen. [SG16-Unicode] Comments on D1629R1 Standard Text Encoding. <http://www.open-std.org/pipermail/unicode/2019-September/000667.html>, 2019.
- [9] ThePhD. sol2. <https://github.com/ThePhD/sol2>, 2019.
- [10] ThePhD. C++ Standardization Efforts. <https://theohd.github.io/portfolio/standard>, 2019.
- [11] ThePhD. Seize the Bits of Production: GSOC 2019 Conclusion. <https://theohd.github.io/seize-bits-production-gsoc-2019>, 2019.
- [12] Tom Honermann et. al. SG16: Unicode Direction. <https://wg21.link/p1238>, 2019.
- [13] Study Group 16 Unicode. SG16 Meeting Notes. <https://github.com/sg16-unicode/sg16-meetings#june-12th-2019>, 2019.
- [14] Ken Whistler. UAX #15: Unicode Normalization Forms. <https://unicode.org/reports/tr15/>, 2019.
- [15] Henri Sivonen. encoding_rs: a web-compatible character encoding library in rust. https://hsivonen.fi/encoding_rs/, 2018.

- [16] IEEE and Open Group. iconv. <https://pubs.opengroup.org/onlinepubs/009695399/functions/iconv.html>, 2004.
- [17] WHATWG. ENcoding Living Standard. <https://encoding.spec.whatwg.org/#the-encoding>, 2019.