std::core_dump

The Newsletter of the C++ Alliance

Issue 2
March 2026

In this Issue

CppCon 2026 Call for Presenters

Consider doing a talk at CppCon 2026! We'd be happy to help you submit a proposal. Whether it's a technical deep-dive, project update, or lessons learned – it's a great opportunity to connect with the community. Reach out to Mark Cooper and we'll start the process.

Presenting the New Era of Networking in Croydon
by Harry Bott

Later next month the ISO C++ standards committee meets in Croydon, just outside London (March 21–28), and Boost and the C++ Alliance will be well represented. We are bringing a focused set of papers on the future of I/O and networking in C++, drawing directly on the decades of real-world experience the Boost community has accumulated through libraries like Asio and Beast. Our goal is to help ensure that the standard's approach to network I/O is informed by what practitioners have learned shipping production code with Boost.

Because these topics cut across so many areas of the standard, Boost and Alliance members will be participating in a number of study groups and working groups where their experience is relevant.

If you are a Boost contributor or maintainer attending Croydon, we would love to connect – whether that's comparing notes on papers, sharing implementation experience, or just putting faces to names. Please reach out and let's make the most of having so many of us in the same place.

More to come after the meeting.

Boost.URL: Constexpr URL Parsing

by Alan de Freitas

Boost.URL now supports constexpr URL parsing in C++20 and later. All parse functions (parse_uri,parse_uri_reference, parse_relative_ref, parse_absolute_uri, and parse_origin_form) are fully constexpr, enabling URLs to be parsed and validated entirely at compile time. A malformed URL literal becomes a compile error rather than a runtime failure, so bugs in hardcoded endpoints, API base paths, and route patterns are caught before the program ever runs.

```
// Parsed and validated at compile time.

// A malformed literal would fail to compile.

constexpr url_view api_base =

      parse_uri("https://api.example.com/v2").value();
```

Pre-parsed constexpr URL views also serve as zero-cost constants: because all parsing happens during compilation, components like scheme, host, and port are available at runtime with no parsing overhead. This is useful for applications that compare against well-known endpoints, pre-populate configuration defaults, or build routing tables without paying for string parsing at startup.

Making this work required coordination across three Boost libraries. Boost.URL's parsing depends on Boost.System for error codes and Boost. Optional (via `result<T>`) for return types, and both libraries needed constexpr improvements to make this possible. Thanks to Peter Dimov for extending constexpr support in Boost.System and Andrzej Krzemienski for doing the same in Boost.Optional. This feature wouldn't have happened without that collaboration.

Boost Docs Localization Pipeline

by Will Pak

Boost is used worldwide, but Boost documentation is still *English-only*, which can be a barrier for new users and teams who do their day-to-day work in other languages. Over the past few months we've been laying the groundwork to change that in a way that scales to Boost's size without slowing releases or adding an ongoing maintenance burden for library authors. This is still a work in progress. The goal is to make Boost docs available in multiple major languages over time, starting with a small set where demand and translator capacity are strongest, and expanding as participation grows.

We started by piloting the workflow on a couple of libraries (`Boost.JSON` and `Boost.Unordered`) and standing up a customized `Weblate` instance so translation work can happen in a structured, reviewable process. In parallel, we built the foundational plumbing: a Boost-like repository structure and CI automation to keep translation repos synchronized with upstream docs. This is still early-stage; the workflow is taking shape, but it's not yet polished for broad participation. The good news is that we now have a concrete form to iterate on, harden, and eventually open up for mass participation. We'd love help both in evolving the tooling/workflow itself and in the ongoing translation and review effort.

Links:
- Boost Translation Website - https://boost-weblate.cloud/
- Website Source Code - https://github.com/CppDigest/weblate/tree/dev
- Translated Boost Documentation - https://github.com/CppDigest/boost-docs-translation/tree/local-zh_Hans

Introducing the Boost Blueprint Series on X

by Mark Cooper

We've recently launched a new educational initiative on X called the Boost Blueprint. The goal is to establish our @Boost_Libraries account on X as a center of gravity for learning modern C++. By focusing on practical, bite-sized code tips – blueprints – we are showing developers how to solve modern engineering challenges using Boost's peer-reviewed libraries.

Core Series Features

Visual code snippets: high-contrast, "Carbon-style" code blocks optimized for mobile
Factual problem-solving: each post leads with a technical "hook" (a common bottleneck) followed by a reply tweet "payoff" (the Boost Blueprint solution)
Modern standards: a focus on C++20/23 integration, such as using co_await with Boost.Asio or high-performance metaprogramming with Boost.Mp11

Early Success & Results

While the series is only in its first week, the response from the C++ community has been great. We're seeing a significant lift in engagement (replies, bookmarks and reposts) and new followers. The goal is to provide a little education to the community every day. Please support the series by liking, replying and reposting!

Boost Library Updates

The Boost community continues to advance C++ development with significant library and infrastructure updates. Here's a roundup of recent progress across several key libraries.

Hub Proposal Scheduled for Review

Author: Joaquín M López Muñoz

Hub (proposed as part of Boost.Container with the name `boost::container::hub`) is a near drop-in replacement of the upcoming `std::hive` container that uses a more compact data structure for excellent performance. The official review for acceptance in Boost will start on April 16.
https://github.com/joaquintides/hub

Boost.Container, Boost.Intrusive and Boost.Interprocess

Author/Maintainer: Ion Gaztañaga

Boost.Container receives a new "segtor" container, an optimized, single-ended brother of classic "deque".

Boost.Intrusive implements new methods inspired on C++20 and newer standard additions, and optimized.

Boost.Interprocess simplifies usage and ergonomics, allowing it to automatically propagate the allocator in many cases following the standard C++ 'uses-allocator' pattern.

Code Coverage Tool for Boost Libraries

Authors: Sam Darwin, Julio Estrada

Sam and Julio have added a tool to the Boost.CI repo that makes it trivial to add gcovr-based code coverage analysis to Boost libraries and see the results nicely displayed as a GitHub Page. The tool is still work in progress but perfectly functional and usable at this stage.

https://github.com/boostorg/boost-ci/blob/master/docs/code-coverage.md

Jan website traffic

48k
uniques

Total repositories using Boost libraries

(repos with 10+ stars)

13,712

X followers

2,005

New Boost Testimonial Partners

AutoDock Vina
https://www.boost.org/testimonials/oleg_trott/

ArrayFire (in progress)
https://arrayfire.com/

Space Communications Cognitive Engine by Penn State & NASA (in progress)
https://ntrs.nasa.gov/api/citations/20170007960/downloads/20170007960.pdf