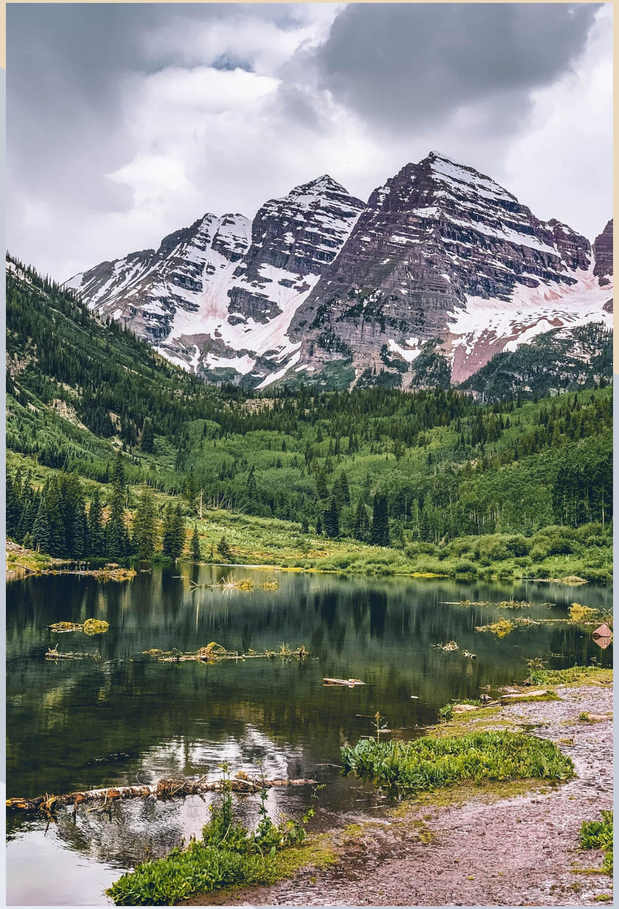


The Boost mission is threefold:

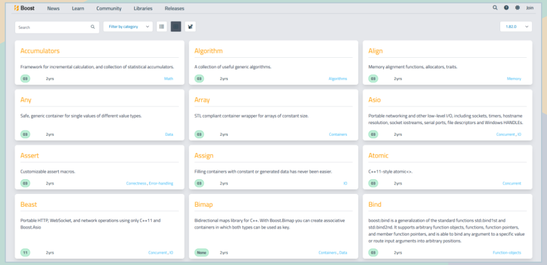
- Develop high-quality, expert-reviewed, open-source C++ libraries
- Incubate C++ standard library enhancements
- Advance and disseminate C++ development best practices

Boost is guided by our shared values of transparency, inclusivity, consensus-building, federated authorship, and community-driven leadership.

New website design

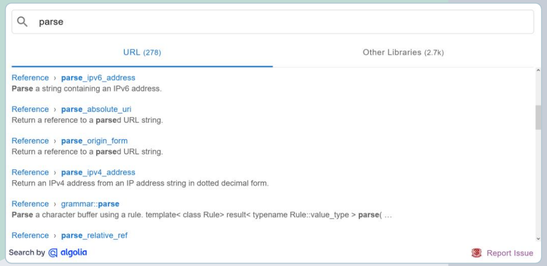
In progress at preview.boost.org

boost.org is getting its first revamp in 25 years! In addition to cooler looks, the new website will offer user log-in with GitHub or Google credentials, web-based forums tightly integrated with the classical mailing lists, updated tutorials, news section, and more!



Integrated full-text search

The embedded widget indexes documentation from all libraries to provide comprehensive, precise and structured search beyond what's possible with external engines.



Modularization

A recurrent request from users is the possibility to install individual Boost libraries to minimize HD space and build times. This is already provided by **vcpkg** and we're working with **Conan** to get there too.

Behind the scenes, a lot of work is underway to support modular builds by major package managers:

- Circular dependencies between Boost libs have been eliminated since 1.77 (Aug 2021).
- The **boostdep** tool automatically extracts the internal dependencies of any library.
- Lib repos are being updated so that **B2**, the official Boost builder, can work with them separately without requiring a full superproject download.

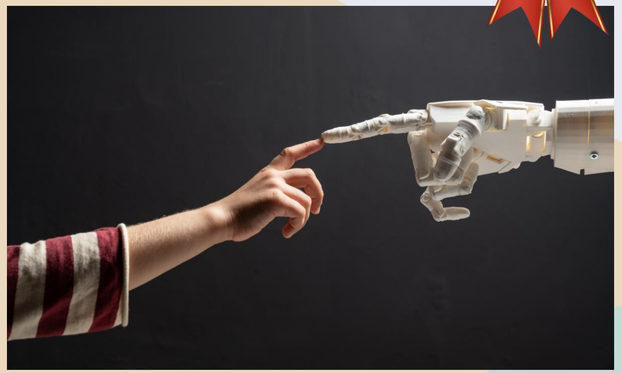
Stay tuned for further advances in modular installs!

CMake support

In development

Git clone Boost and use standard CMake procedures to build, install and use it. Integration in your builds is supported via **find_package**, **add_subdirectory** or **FetchContent**, either for the entire Boost super-project or for individually specified libraries.

Boost



The most widely used collection of high quality, peer-reviewed C++ libraries on the planet.

The libraries continue to thrive, delivering unparalleled performance and reliability.

- New website
- Full-text search
- Smaller installs
- C++03 deprecation
- Discussions groups at cpplang.slack.com

www.boost.org

Boost.Mp11

Fast, useful metaprogramming for the everyday programmer. Based on C++11 template aliases and variadic templates, **Boost.Mp11** makes it easy to automatically generate test cases for generic code, manipulate type lists as comfortably as run-time containers, or, in combination with **Boost.Describe**, visit classes for serialization or JSON conversion.

Boost.Describe

You don't have to wait for the next decade to enjoy C++ static reflection capabilities. Annotate your types with simple **BOOST_DESCRIBE_*** macros to automatically get a wealth of type information at compile time: member types and signatures, names, base classes… Best used in combination with **Boost.Mp11**.

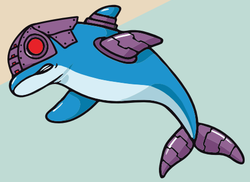
```cpp
#include <boost/describe.hpp>
#include <boost/mp11.hpp>
#include <cstdio>

enum E
{
  v1 = 11, v2, v3 = 5
};

BOOST_DESCRIBE_ENUM(E, v1, v2, v3)

int main()
{
  boost::mp11::mp_for_each<
    boost::describe::describe_enumerators<E>>([](auto D){
    std::printf(
      "%s: %d\n", D.name, D.value );
  });
}
```

Boost.MySQL



Access your **MySQL** and **MariaDB** servers in a truly asynchronous fashion, using **Boost.Asio**'s async model.

Featuring a static interface that allows parsing rows into your own data structures, and a dynamic interface for maximum flexibility.

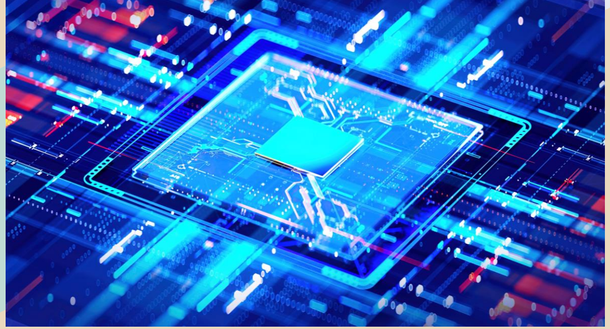
All of this with zero dependencies on the official C drivers!

Boost.Redis

Coming in Boost 1.84 (Dec 2023)

High-level client library for **Redis 6** or higher based on **Boost.Asio**.

Designed with a focus on performance, **Boost.Redis** allows for concurrent requests, pipelining and in-place data retrieval. STL containers and user-defined types can be easily managed through string serialization.



Boost.Unordered

In addition to standards-compliant implementations of std::unordered_map and relatives, **Boost.Unordered** now features:

boost::unordered_flat_map: One of the fastest hashmaps in the market. Takes advantage of open-addressing techniques and SIMD instructions to easily outperform standard containers by 3x or more.

boost::concurrent_flat_map: Suitable for high-load concurrent scenarios. To avoid deadlocking issues with iterators, this container introduces a novel, iterator-free API based on the visitation paradigm.

Boost.URL

Full implementation of **RFC3986** URI/URL specification.

Parse URLs into their constituent parts and use the container-based API to analyze them and synthesize new ones with ease.

Candidate Boost.Charconv

Upcoming review

Use <charconv> in your C++11 projects **today!** Convert from a sequence of characters to any integral/floating point type and back, with maximum speed and precision.

```cpp
#include <boost/charconv.hpp>

const char* buffer = "42";
int v = 0;
auto r = boost::charconv::from_chars(
           buffer,
           buffer + std::strlen(buffer), v);
assert(r.ec == std::errc());
assert(v == 42);

char buffer[64];
int v = 123456;
auto r = boost::charconv::to_chars(
           buffer,
           buffer + sizeof(buffer) - 1, v);
assert(r.ec == std::errc());
// Strncmp returns 0 on match
assert(!strncmp(buffer, "123456", 6));
```