

17 August 2017

Blog post

- ▶ Accessibility improvements
- ▶ [C++17 Features And STL Fixes In VS 2017 15.3](#) by STL
 - ▶ The STL now uses C++14 constexpr unconditionally, instead of conditionally-defined macros.
 - ▶ The STL now uses alias templates internally.
 - ▶ The STL now uses `std::move()` internally, instead of stylistically misusing `std::forward()`.
 - ▶ The STL no longer marks functions as `__declspec(dllimport)`. Modern linker technology no longer requires this (???)

How a blind developer uses Visual Studio

YouTube

Passing parameters to constructors

```
1 class Foo {  
2 public:  
3     Foo(Bar bar, Baz baz)  
4         : bar_(std::move(bar))  
5           , baz_(std::move(baz))  
6         {}  
7 };
```

Is this better than using const references?

C++ Core Guidelines: [passing consume parameters](#)

p0052r5 - Generic Scope Guard and RAII Wrapper for the Standard Library

PDF

```
1 auto scope_exit = make_scope_exit([&]{ /* always */ });  
2 auto scope_success = make_scope_success([&]{ /* successful scope exit */ });  
3 auto scope_fail = make_scope_fail([&]{ /* exception */ });
```

Compare with GSL's finally [Link](#)

PDF

We propose a `basic_osyncstream`, that buffers output operations for a wrapped stream. The `basic_osyncstream` will atomically transfer the contents of an internal stream buffer to a `basic_ostream`'s stream buffer on destruction of the `basic_osyncstream`.

Example:

```
1 | osyncstream{cout} << "The answer is " << 6*7 << endl;
```

Familiar template syntax for generic lambdas

PDF

C++14:

```
1 [](auto x) { /* ... */ }
```

Proposed:

```
1 []<typename T>(T x) { /* ... */ }  
2 []<typename T>(T* p) { /* ... */ }  
3 []<typename T, int N>(T (&a)[N]) { /* ... */ }
```


[Link](#)

Feedback, edits/diffs, discussions.

Post

- ▶ *System error support in C++0x* by Chris Kohlhoff: [Part 1](#), [Part 2](#), [Part 3](#), [Part 4](#), [Part 5](#)
- ▶ `std::error_code`

DIMWITS: DIMensional analysis With unITS

GitHub - C++14, Copyright (c) 2016, Los Alamos National Security, LLC

```
1  /* quantities play nicely with auto */
2  auto myVelocity = 1.0 * meter / second;
3  std::cout << "The speed is: " << myVelocity << std::endl;
4
5  /* quantities of the same dimensionality can be implicitly converted */
6  Quantity<Foot> myLength = 1.0 * meter;
7  std::cout << "1 meter in feet is: " << myLength << std::endl;
8
9  /* si-prefixes can be specified on either side of the assignment */
10 Quantity<Kilo<Gram>> myMass = 1.0 * mega(tonne);
11 std::cout << "1 megatonne in kilograms is: " << myMass << std::endl;
12
13 /* NIST values for common physical constants are provided */
14 std::cout << "The speed of light is: " << constant::lightSpeed << std::endl;
```

sltbench: a C++ micro-benchmarking tool

GitHub – Apache 2.0 licence

► 4.7x times faster than googlebench

```
1 void my_function()
2 {
3     std::vector<size_t> v(100000, 0);
4     std::sort(v.begin(), v.end());
5 }
6
7 SLTBENCH_FUNCTION(my_function);
8
9 SLTBENCH_MAIN();
```

spdlog: an ultra-fast C++ logging library

GitHub - C++11, MIT licence

- ▶ header-only
- ▶ Linux, FreeBSD, Solaris, Mac OS, Windows, Android
- ▶ Uses **{fmt}** library for formatting
- ▶ Async mode using lock-free queues
- ▶ Custom formatting
- ▶ Conditional logging
- ▶ Targets: rotating/daily log files, console (w/colour), syslog, Windows debugger
- ▶ Severity-based filtering

GitHub - C++11, Boost Licence

- ▶ Based on libc++ implementation of `std::variant` (same author)
- ▶ Continuously tested against libc++'s `std::variant` test suite
- ▶ Single-header
- ▶ [Documentation](#)

- ▶ [Meetup](#)
- ▶ [YouTube](#)

Wt: a C++ web toolkit

- ▶ [Home page](#)
- ▶ Licence: GPL + commercial
- ▶ [GitHub](#)

Beast is now a part of Boost

- ▶ Boost.ASIO-based HTTP and WebSockets library
- ▶ Header-only
- ▶ Version 100!
- ▶ Uses callbacks or coroutines
- ▶ C++11
- ▶ Dependencies: Boost.System, Boost.Coroutine (optional)
- ▶ [GitHub](#)
- ▶ [Reddit thread](#)

YouTube Playlist

Towards a Good Future

P0676R0, [GitHub](#)

- ▶ Based on [Adobe stlab's future](#) and [Bloomberg dplp promise](#)
- ▶ Author recommends against adoption of the proposed `std::future` extensions in Concurrency TS
- ▶ `std::future` is crippled
- ▶ Futures need to be copyable (currently limited to just one `.then()`)

```
1 future<int> a;  
2 a.then([](int x){ /* do something */ });  
3 a.then([](int x){ /* also do something else. */ })
```

- ▶ Futures need to be cancellable