`unique_function` vs. `any_invokable` - Bikeshedding Off the Rails

Document number: P1737R0
Date: 2019-06-17
Project: Programming Language C++, Library Evolution Working Group
Reply-to: Nevin "☺" Liber, nliber@anl.gov

Table of Contents

Introduction

Why are we renaming `unique_function`? `any_invocable` is a name only an expert committee member could love.

Motivation and Scope

Six years ago the idea for the fundamental vocabulary type `unique_function` came up on std::discussion. While the name was never disputed on the public lists, over the years LEWG has spent time bikeshedding it, the latest incarnation being `any_invokable`.

Just this past week, the following unsolicited conversation happened in the #general channel on Cpplang.slack.com. Here are the highlights (names have been changed; full discussion attached):

Alpha
speaking of std::function
why has unique_function been renamed to any_invocable?
or are those two different things i'm confusing

Bravo
Someone's trying to introduce a new naming convention for polymorphic wrappers

Alpha
it seems utterly inconsistent
:disappointed:

[....]

Hotel
"Any invocable" makes me think of a hybrid of std::any and std::function.
I wouldn't expect it to be move-only from the name

[...]

Bravo
Look, `std::function` is a polymorphic wrapper for Invokable and CopyConstructible. `std::any` is a polymorphic wrapper for CopyConstructible
Given ___ which is a polymorphic wrapper for Invokable and MoveConstructible, how do we get to `any_invokable`?

[...]

Delta

"any invocable" makes _some_ sense in a world in which `std::any` is named "any copyable" and `std::function` is named "any copyable invocable"

Design Decisions

To summarize the Kona 2019 discussion, LEWG considers type erasure to be the most important property of this, and because this property is so important, it should be reflected in the name, and that name should start with `any_`.

I strongly disagree that being type erased is an important, let alone the most important property. It is an implementation detail. The two important properties are that it is callable and that it is move-only.

And if the trend is that `any` should indicate type erased types, where are the proposals for these other type erased types? SD-4 specifically says that "We cannot act on ideas without papers". Now, some leeway may be given when naming something more general than a specific type (such as `view` for read-only views, `span` for modifiable views, etc.), but given that we are only on our 4th type erased holder in the standard in over two decades, it behooves us to see if this is really a trend.

Plus, the naming convention of `any_` did not go through a paper, yet it seems like LEWG has already decided and applied this. We need to take a step back.

`any` implies more than just type erased (for instance, it is copyable and it holds a copyable type), and `any` isn't the only type erased holder in the standard. Comparing type erased holders and move-only types in the standard we get:

	unique_ptr<T>	shared_ptr<T>	any<T>	reference_wrapper<T>	function<T>	*any_invokable<T>*
Type Erased		√	√		√	√
Move-only	√					√
T Moveable						√
Copyable		√	√	√	√	
T Copyable			√		√	
Callable				√	√	√
User friendly name	√	√	√	√	√	
User hostile name						√

`any_invocable` has as much in common with any as it does with `unique_ptr`, and it has *more* in common with `function`. Yet the name reflects neither of these.

Speaking of `function`, we also have the proposed `function_ref` for a non-owning reference to a callable. The name `unique_function` fits well with those, while `any_invocable` does not.

any_invocable is still a misnomer, because it is both move-only itself and there is (currently) a movable requirement on T[1], so it cannot store any old invocable. Invocable is even hard to spell (c or k? One or two es?).

It has been suggested that "If you want to participate in naming, sit in LEWG". That is just not practical as no one knows when LEWG will suddenly partake in a naming discussion.

It has also been suggested that "P0228 proceed to wording and LWG review with the current name from Kona, and that name change proposals come in as a separate LEWG paper." The bar is much, much higher to change the name once it is in the Working Draft. In all likelihood this will replace most uses of function as a fundamental vocabulary type, and some people may vote against putting it into the standard at all if the name is unacceptable.

Naming is very hard. Naming is extremely important. Naming fundamental vocabulary types even more so. Yet the way we bikeshed names is horribly, horribly broken. Given that names stick around practically forever, we shouldn't be naming things on the fly at meetings. Name changes, like everything else, should have to go through papers, so there is at least a chance that we would have thought about the name for more than the time it takes to list it in a poll, and non-LEWG regulars would have a chance to provide feedback without having to spend 100% of their time in LEWG on the off-chance LEWG will decide to rename something.

I urge this committee to go back to naming this fundamental vocabulary type unique_function.

[1] We could relax the movable requirement on T by adding an in_place_type_t<T> constructor similar to the one in variant. That would still leave the requirement that unique_function itself is move-only.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative. Additionally, this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

I would like to thank Geoffrey Romer & Zheming Jin for proofreading this and making suggestions, and the various participants (far too numerous to mention) in LEWG, LWG, and Cpplang Slack #general for helping me to solidify my points.

References

N4810 Working Draft, Standard for Programming Language C++
P0228 unique_function: a move-only std::function
P0792 function_ref: a non-owning reference to a Callable

June 7th, 2019 #general discussion on Cpplang.slack.com https://cpplang.slack.com/archives/C21PKDHSL/p1559932245189600 (names have been changed):

Alpha [1:30 PM]
speaking of std::function
why has unique_function been renamed to any_invocable? (edited)
or are those two different things i'm confusing

Bravo [1:31 PM]
Someone's trying to introduce a new naming convention for polymorphic wrappers

Alpha [1:31 PM]
it seems utterly inconsistent
:disappointed:

Charlie [1:50 PM]
@Delta I suppose, just doesn't quite feel right (edited)

Delta [1:53 PM]
it's a bit clunky, but it is explicitly supported for that kind of use case

nevin [2:32 PM]
It's wrong, and hopefully I can finish up my paper on it. Mind if I quote you @Alpha?

Alpha [2:33 PM]
not at all

Delta[2:43 PM]
@nevin what's the name your paper uses?

nevin [2:43 PM]
unique_function

Echo [3:04 PM]
is it too much ask for consistensy? `unique_ptr`, `unique_lock` and therefore `any_invocable` makes sense?

Foxtrot [3:05 PM]
Well tbf
"any invocable" includes the move-only ones

Bravo [3:05 PM]
`unique_ptr` is not a polymorphic wrapper around a moveable pointer

Golf [3:05 PM]
You can also have lots of unique_functions pointing to the same function

Foxtrot [3:06 PM]
Not very unique, now is it?

Bravo [3:06 PM]
New feature request: `unique_function` should add a random side effect to every input, to ensure uniqueness

Foxtrot [3:06 PM]
You know, "any invocable" does make more sense now that I think about it out loud

Golf [3:06 PM]
std::unique doesn't make every range contain only unique values either though... requires a sorted range for that
I am going to try my hardest to not be in the room when this naming battle happens though
should be easy if it happens in Cologne... I'll be on the other side of an ocean

Hotel [3:08 PM]
"Any invocable" makes me think of a hybrid of std::any and std::function.
I wouldn't expect it to be move-only from the name

Foxtrot [3:09 PM]
`std::any_movable_invocable`

Golf [3:09 PM]
`unique` has accidentally picked up the idea of meaning `move only`

Bravo [3:10 PM]
Look, `std::function` is a polymorphic wrapper for Invokable and CopyConstructible. `std::any` is a polymorphic wrapper for CopyConstructible
Given ___ which is a polymorphic wrapper for Invokable and MoveConstructible, how do we get to `any_invokable`?

Foxtrot [3:10 PM]
Huh

Delta [3:10 PM]
"any invocable" makes _some_ sense in a world in which `std::any` is named "any copyable" and `std::function` is named "any copyable invocable"
(destructible is always assumed)

Bravo [3:11 PM]
Is MoveConstructible assumed here?

Delta [3:11 PM]
no

Bravo [3:11 PM]
I guess a wrapper can't really wrap unmoveable types

Delta [3:11 PM]

it could if, for instance, it constructs it in place

Bravo [3:11 PM]
Good point

Delta [3:12 PM]
we'll have to see the actual proposed wording to figure out how much movement the implementation happens to _require_

Charlie [3:12 PM]
the advantage of dictatorship languages is that someone has a single view so there tends to be at least a little more consistency
when you have a committee different views (on everything, including naming) are always waxing and waning
@Bravo unique_ptr is a wrapper type, no? You can have a unique_ptr<mutex> no problem.

India [3:13 PM]
The general meaning of wrapper type is in place wrapper.

Charlie [3:14 PM]
I've never heard of this before

India [3:14 PM]
And you can wrap a non-movable type as long as the wrapper type is also non-movable.

Juliett [3:14 PM]
does anyone use folly::Poly here?

Charlie [3:14 PM]
std::function is only sometimes a wrapper because it's only sometimes in place?

India [3:14 PM]
@Charlieoptional is a wrapper type.
function is not a wrapper type.

Delta [3:14 PM]
`std::function` is a function call wrapper, it is always a wrapper, it might be a different meaning of wrapper

Charlie [3:15 PM]
Curious where is this definition of wrapper coming from? Any source?

Bravo [3:15 PM]
But is `function` a container? :smirk:

Delta [3:15 PM]
sorry, callable wrapper

Juliett [3:15 PM]
is optional a monad?

Kilo [3:15 PM]
In C++? No

Lima [3:15 PM]
yes, it's the maybe monad in haskell

Juliett [3:15 PM]
no its not, not on its own, right? what are the two operations

Juliett [3:15 PM]
a monad is a triple as I understood it

India [3:15 PM]
@Deltayeah, callable wrapper makes sense. It's like wrapping a function pointer.

Kilo [3:16 PM]
@Juliett Correct

Charlie [3:16 PM]
Yes, std::function is a callable wrapper. So, it is a wrapper. Even though it's not (always) in place.

Delta [3:16 PM]
it's actually "call wrapper", I got it partially right twice
http://eel.is/c++draft/func.wrap.func#3