# Thoughts on creating a tracking pointer class, part 8: Tracking const objects

devblogs.microsoft.com/oldnewthing/20250820-00/?p=111490

August 20, 2025

Last time, we added the ability to create non-modifying tracking pointers, but it required that you start with a non-const trackable object. But what if you want to create a non-modifying tracking pointer to an object to which you have only a const reference?

It took me a few tries before I hit upon the simple solution.[1]

We just need to make the trackable object's `m_trackers` mutable, and then patch up the consequences.

```
template<typename T>
struct trackable_object
{
    ⟦ ... ⟧

    tracking_ptr<T> track() noexcept {
        return { owner() };
    }

    tracking_ptr<const T> track() const noexcept {
        return { owner() };
    }

    tracking_ptr<const T> ctrack() const noexcept {
        return { owner() };
    }

private:
    friend struct tracking_ptr_base<T>;

    T* owner() const noexcept {
        return const_cast<T*>(static_cast<const T*>(this));
    }

    tracking_node mutable m_trackers;

    ⟦ ... ⟧
};
```

After making the `m_trackers` mutable, we can make the `ctrack()` method `const`. We may as well also add a `track() const` that produces a read-only tracker from a const object. This is analogous to how C++ standard library container `begin()` and and `end()` methods produce read-only iterators if obtained from const containers.

Casting away const when creating and updating the `tracking_ptr<const T>` is okay because the `tracking_ptr<const T>`'s `get()` method will reapply const before giving it to the client, and the only other use of the pointer is to access the `m_trackers`, which is now mutable.

There's still a problem: Although you can convert a `T*` to a `const T*`, a `std::unique_ptr<T>` to a `std::unique_ptr<const T>`, and a `std::shared_ptr<T>` to a `std::shared_ptr<const T>`, you cannot convert a `tracking_ptr<T>` to a `tracking_ptr<const T>`. We'll fix that next time.

**Bonus chatter**: I considered whether I should also support `tracking_ptr<volatile T>` and `tracking_ptr<const volatile T>`. It wouldn't be hard, but it would be extra typing. I decided not to, on the grounds that the C++ standard library typically doesn't bother with `volatile` either: Standard containers have `begin()` and `cbegin()` but not `vbegin()` or `cvbegin()`.

[1] As Blaise Pascal is reported to have written, “If I had more time, I would have written a shorter letter.” Sometimes it takes a lot of work to come up with what ends up looking easy.