Thoughts on creating a tracking pointer class, part 9: Conversion

devblogs.microsoft.com/oldnewthing/20250821-00/?p=111492

August 21, 2025

Last time, we added the ability to create tracking pointers from const objects. But we discovered that you could not convert a `tracking_ptr<T>` to a `tracking_ptr<const T>`. But this should be possible, because the available operations on a tracking pointer to a const object is a subset of the operations available to a tracking pointer to a non-const object.

We can perform the conversion by copying/moving the underlying `tracking_ptr_base`.

```
template<typename T>
struct tracking_ptr : tracking_ptr_base<std::remove_cv_t<T>>
{
private:
    using base = tracking_ptr_base<std::remove_cv_t<T>>;

public:
    T* get() const { return this->tracked; }

    using base::base;
    tracking_ptr(base const& other) : base(other) {}
    tracking_ptr(base&& other) : base(std::move(other)) {}
};
```

If somebody has a `tracking_ptr<T>` it converts to a `tracking_ptr<const T>` by means of our two new constructors, which copy/move the common `tracking_ptr_base<T>`, which is the thing that babysits the `shared_ptr<T*>`. To avoid repetitive typing, we make `base` an alias for the base class `tracking_ptr_base<std::remove_cv_t<T>>`.

We don’t need to write custom assignment operators because assignment can be performed by converting the `tracking_ptr<T>` to a `tracking_ptr<const T>`, and then assigning the `tracking_ptr<const T>`.

But wait, there’s a problem with this. We’ll look at it next time.