C++/WinRT implementation inheritance: Notes on winrt::implements, part 6

devblogs.microsoft.com/oldnewthing/20250226-00

February 26, 2025

Last time, we were looking for a way to allow a `winrt::implements`-based base class to defer implementation of a method to its derived class. The problem is that if you use `winrt::implements` to implement an interface, you have to implement all the methods. But you might also want to leave some of the methods for the derived class to implement in a manner of its choosing (for example, choosing whether the parameters are references or values, or allowing the parameters to be templated), which means that the virtual method approach doesn't work. Something has to give.

There are multiple solutions, depending on which requirement you want to weaken.

You can weaken the "leave some of the methods for the derived class to implement" requirement by implementing it yourself but forwarding the call to the derived class via CRTP.

```
// A simple copy drop target provides no custom feedback
// and accepts anything by copy.
template<typename D>
struct SimpleCopyDropTarget :
    winrt::implements<SimpleCopyDropTarget<D>,
    winrt::ICoreDropOperationTarget>
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        EnterAsync(winrt::CoreDragInfo const& info,
                   winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncOperation<winrt::DataPackageOperation>
        OverAsync(winrt::CoreDragInfo const& info,
                  winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncAction
        LeaveAsync(winrt::CoreDragInfo const&)
    {
        co_return;
    }

    winrt::IAsyncOperation<winrt::DataPackageOperation>
        DropAsync(winrt::CoreDragInfo const& info)
    {
        return static_cast<D*>(this)->
            DropAsyncImpl(info);
    }

protected:
    winrt::DataPackageOperation GetOperation(
        winrt::CoreDragInfo const& info)
    {
        return info.AllowedOperations() &
               winrt::DataPackageOperation::Copy;
    }
};

struct Derived : winrt::implements<
    Derived,
    SimpleCopyDropTarget<Derived>>
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        DropAsyncImpl(winrt::CoreDragInfo info)
    {
        auto lifetime = get_strong();
```

```
        auto operation = GetOperation(info);
        if (!(operation & winrt::DataPackageOperation::Copy)) {
            co_return winrt::DataPackageOperation::None;
        }

        ⟦ process the drop ⟧

        co_return winrt::DataPackageOperation::Copy;
    }
};
```

We implement `DropAsync` in the base class but immediately forward the call out to the derived class's `DropAsyncImpl` method. This is done via CRTP so that the derived class has full flexibility in deciding how to accept the parameters.

If you have access to "deducing this", then you can let the "this" deduction do the work instead of CRTP.

```
// A simple copy drop target provides no custom feedback
// and accepts anything by copy.
struct SimpleCopyDropTarget :
    winrt::implements<SimpleCopyDropTarget,
    winrt::ICoreDropOperationTarget>
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        EnterAsync(winrt::CoreDragInfo const& info,
                   winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncOperation<winrt::DataPackageOperation>
        OverAsync(winrt::CoreDragInfo const& info,
                  winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncAction
        LeaveAsync(winrt::CoreDragInfo const&)
    {
        co_return;
    }

    winrt::IAsyncOperation<winrt::DataPackageOperation>
        DropAsync(this auto&& self,
                  winrt::CoreDragInfo const& info)
    {
        return self.DropAsyncImpl(info);
    }

protected:
    winrt::DataPackageOperation GetOperation(
        winrt::CoreDragInfo const& info)
    {
        return info.AllowedOperations() &
               winrt::DataPackageOperation::Copy;
    }
};

struct Derived : winrt::implements<
    Derived,
    SimpleCopyDropTarget>
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        DropAsyncImpl(winrt::CoreDragInfo info)
    {
        auto lifetime = get_strong();

        auto operation = GetOperation(info);
```

```
        if (!(operation & winrt::DataPackageOperation::Copy)) {
            co_return winrt::DataPackageOperation::None;
        }

        ⟦ process the drop ⟧

        co_return winrt::DataPackageOperation::Copy;
    }
};
```

Another option is to weaken the “implement an interface” part of “use `winrt::implements` to implement an interface”. We can simply omit `ICoreDropOperationTarget` from the list of interfaces implemented by the base class, since our base class doesn’t contain a full implementation. Instead, let the derived class finish the implementation and declare the interface there.

```
// A simple copy drop target provides no custom feedback
// and accepts anything by copy.
struct SimpleCopyDropTarget :
    winrt::implements<SimpleCopyDropTarget,
    winrt::IInspectable> // no ICoreDropOperationTarget
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        EnterAsync(winrt::CoreDragInfo const& info,
                   winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncOperation<winrt::DataPackageOperation>
        OverAsync(winrt::CoreDragInfo const& info,
                  winrt::CoreDragUIOverride const&)
    {
        co_return GetOperation(info);
    }

    winrt::IAsyncAction
        LeaveAsync(winrt::CoreDragInfo const&)
    {
        co_return;
    }

    // DropAsync must be implemented by derived class

protected:
    winrt::DataPackageOperation GetOperation(
        winrt::CoreDragInfo const& info)
    {
        return info.AllowedOperations() &
               winrt::DataPackageOperation::Copy;
    }
};

struct Derived : winrt::implements<
    Derived,
    SimpleCopyDropTarget,
    winrt::ICoreDropOperationTarget>
{
    winrt::IAsyncOperation<winrt::DataPackageOperation>
        DropAsync(winrt::CoreDragInfo info)
    {
        auto lifetime = get_strong();

        auto operation = GetOperation(info);
        if (!(operation & winrt::DataPackageOperation::Copy)) {
            co_return winrt::DataPackageOperation::None;
        }
```

```
        ⟦ process the drop ⟧

        co_return winrt::DataPackageOperation::Copy;
    }
};
```

Our `SimpleCopyDropTarget` no longer implements `ICoreDropOperationTarget`. Instead, it is `Derived` which implements `ICoreDropOperationTarget`. The `SimpleCopyDropTarget` happens to provide some really handy implementations of `ICoreDropOperationTarget` methods, but they aren't actually hooked up to the `ICoreDropOperationTarget` interface until the `Derived` class says, "And I implement `ICoreDropOperationTarget`."

The upside of this is that you don't have to use CRTP or forwarders. The downside of this is that the `Derived` class has to remember to say "And I implement `ICoreDropOperation-Target`," because if nobody says it, then the interface isn't implemented by anybody!

But wait, the `SimpleCopyDropTarget` doesn't implement anything interesting any more. Why are we even bothering?

Next time, we'll solve the problem a third way: Don't even bother.