C++/WinRT implementation inheritance: Notes on winrt::implements, part 7

devblogs.microsoft.com/oldnewthing/20250227-00

February 27, 2025

Last time, we simplified a base class so that it doesn't fully implement any interfaces at all. So why are we using `winrt::implements` in the first place?

Yeah, why?

Let's not.

```
// A simple copy drop target provides no custom feedback
// and accepts anything by copy.
struct SimpleCopyDropTarget
{
    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;
    }
};
```

This base class implements some of the methods of `ICoreDropOperation`, but it doesn't implement `DropAsync`, so it can't be used to implement the interface. So we don't even pretend to! We don't derive from `winrt::implements` at all. Instead, the derived class just uses this base class to implement some of the interface methods that it declares.

```
struct Derived : winrt::implements<
    Derived,
    winrt::ICoreDropOperationTarget>,
    SimpleCopyDropTarget,
{
    winrt::IAsyncOperation<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;
    }
};
```

In more general cases, the helper class may need to offer extensibility points to the derived class, and we can use the existing mechanisms (virtual methods, CRTP, deducing this) to ask the derived class for help.

Next time, we'll compare the pros and cons of the various options.