C++/WinRT implementation inheritance: Notes on winrt::implements, part 1

devblogs.microsoft.com/oldnewthing/20250219-00

February 19, 2025

The C++/WinRT library provides the `winrt::implements` template type for implementing Windows Runtime objects. The general pattern for it is

```
struct MyThing : winrt::implements<MyThing, ⟦ other stuff ⟧>
{
    ⟦ implementation details ⟧
};
```

You repeat the name of the derived type as the first template parameter because `implements` uses the curiously recurring template pattern (CRTP) to access methods of its derived type.

The “other stuff” can contain the following:

- A C++/WinRT interface such as `winrt::Windows::Foundation::IMemoryBuffer`.
- A classic COM interface such as `IStream`. (Requires that you enable classic COM support.)
- A C++/WinRT runtime class as `winrt::Windows::Foundation::Uri`. (Not recommended.)
- A marker class such as `non_agile`.
- Another `winrt::implements`. (Maximum 1.)

Let’s get the middle item out of the way: Specifying a C++/WinRT runtime class is equivalent to specifying its default interface. This is actually more of a curse than a blessing, for in the case that the C++/WinRT runtime class has multiple interfaces, only the first one is used, and the rest are ignored! This means that when you write

```
struct MockMemoryBuffer : winrt::implements<MockMemoryBuffer,
    winrt::Windows::Foundation::MemoryBuffer>
{
    ⟦ implementation details ⟧
};
```

you are implementing only the `IMemoryBuffer` interface and none of the other interfaces like `IClosable`. It *looks like* you're implementing all of `MemoryBuffer`, but you're not.

So don't write that. It's misleading. Just write

```
struct MockMemoryBuffer : winrt::implements<MockMemoryBuffer,
    winrt::Windows::Foundation::IMemoryBuffer>
{
    ⟦ implementation details ⟧
};
```

and then it's clear that you implemented only one of the required interfaces, and then you'll remember to implement the other one.

```
struct MockMemoryBuffer : winrt::implements<MockMemoryBuffer,
    winrt::Windows::Foundation::IMemoryBuffer,
    winrt::Windows::Foundation::IClosable>
{
    ⟦ implementation details ⟧
};
```

Fortunately, implementing somebody else's runtime class typically happens only when writing mocks, so the problem tends to arise in testing rather than in production.

If you are implementing a runtime class declared in an IDL file, then don't use `winrt::implements` directly. Instead, use the autogenerated T-template, which fills in all of the required interfaces for you.

```
// automatically implements everything declared in the IDL
// need to declare only "bonus" interfaces not listed in the IDL
struct Widget : WidgetT<Widget, ⟦ other stuff ⟧>
{
    ⟦ implementation details ⟧
};
```

Next time, we'll look at how `winrt::implements` combines with C++ class inheritance.

Bonus chatter: How do I know this? I read the C++/WinRT code and figured it out. You too can become an expert in something by just reading the code and figuring it out.