Why am I getting the error message “This is only for weak ref support” when I try to create a C++/WinRT weak reference?

devblogs.microsoft.com/oldnewthing/20220330-00

March 30, 2022

Raymond Chen

A customer tried to obtain a weak reference from within a C++/WinRT class implementation by calling `get_weak()`. This normally works, but for some reason, the customer was getting the error

```
winrt\base.h(7594,47): error C2338: This is only for weak ref support.
winrt\base.h(7593): message : while compiling class template member function
'winrt::impl::IWeakReferenceSource
*winrt::impl::root_implements<D,IWidget>::make_weak_ref(void) noexcept'
        with
        [
            D=Widget
        ]
winrt\base.h(7500): message : see reference to function template instantiation
'winrt::impl::IWeakReferenceSource
*winrt::impl::root_implements<D,IWidget>::make_weak_ref(void) noexcept' being
compiled
        with
        [
            D=Widget
        ]
winrt\base.h(7803): message : see reference to class template instantiation
'winrt::impl::root_implements<D,IWidget>' being compiled
        with
        [
            D=Widget
        ]
Widget.h(15): message : see reference to class template instantiation
```

The message “This is only for weak ref support” is rather enigmatic. It sounds like an internal note from the authors of the C++/WinRT library to their future selves, telling them that they made some sort of internal error.

But let’s see if we can figure out what it’s saying. This is another segment in the “Behind C++/WinRT series” (and the larger topic of decoding template metaprogramming).

The error message comes from make_weak_ref, which is a helper function called from get_weak:

```
static_assert(is_weak_ref_source::value, "This is only for weak ref support.");
```

What is `is_weak_ref_source`? Let's look for it:

```
using is_inspectable =
std::disjunction<std::is_base_of<Windows::Foundation::IInspectable, I>...>;
using is_weak_ref_source = std::conjunction<is_inspectable,
std::negation<std::disjunction<std::is_same<no_weak_ref, I>...>>>;
```

We saw in an earlier installment how to read `std::disjunction`, `std:conjunction`, and `std::negation`.

First, let's untangle `is_inspectable`: It checks each of the implemented interfaces `I` to see if it has `Windows::Foundation::IInspectable` as a base class, and then uses `std::disjunction` to "or" all the results together via a template parameter pack expansion. Therefore, `is_inspectable` is true if any of the implemented interfaces has `Windows::Foundation::IInspectable` as a base class, which means that at least one of the interfaces is or is derived from `IInspectable`.

Next, we calculate `is_weak_ref_source`. The parameter pack expansion checks whether any of the implemented interfaces is the marker type `no_weak_ref`. Therefore, through the power of *reading code*, we determine that in order for a type to support weak references, it must

- Implement `IInspectable`, and
- Not mention `no_weak_ref`.

Applying our newfound knowledge, we can see from the error message that the thing being implemented is `root_implements<Widget, IWidget>`. The sole interface is `IWidget`, and there is no mention of `no_weak_ref`, so the force of logic compels us to conclude that `IWidget` must not derive from `IInspectable`.

Checking with the customer confirms that `IWidget` is a classic COM interface and C++/WinRT therefore does not generate weak reference support for `Widget`, which explains why `get_weak()` was producing a compiler error.

Bonus chatter: C++/WinRT pull request 1104 removes the requirement that one of the interfaces derive from `IInspectable`. With that change (available in build 2.0.220224.4), everything supports weak references by default. To remove weak reference support, you say `no_weak_ref`.

Raymond Chen

Follow