How well does C++/WinRT com_ptr support class template argument deduction (CTAD)?

devblogs.microsoft.com/oldnewthing/20240319-00

March 19, 2024

Raymond Chen

Continuing our investigation of which C++ COM wrappers support class template argument deduction (CTAD), next up is the C++/WinRT `winrt::com_ptr`.

This one is easy: C++/WinRT's `com_ptr` doesn't support CTAD, and it doesn't even try. There is no `com_ptr<T>` constructor that takes a `T*`. If you want a `com_ptr` to copy an existing pointer, you call `com_ptr::copy_from`:

```
IWidget* p;

winrt::com_ptr<IWidget> smart;
smart.copy_from(p);
```

C++/WinRT eschews the constructor that makes a copy of an inbound pointer on the theory that it's unclear whether it is taking ownership of the pointer or merely sharing it.

The library author could have provided a deduction guide for the "take ownership" constructor:

```
template<typename T>
com_ptr(T*, take_ownership_from_abi_t) -> com_ptr<T>;
```

As a consumer, though, you shouldn't be creating deduction guides for somebody else's classes. Instead, you can use a maker function.

```
template<typename T>
winrt::com_ptr<
    std::enable_if_t<std::<is_base_of_v<::IUnknown, T>, T>>
    make_com_ptr(T* p, winrt::take_ownership_from_abi_t)
{
    return { p, winrt::take_ownership_from_abi };
}
```

There is some extra magic in the return value to activate the function only if `T` derives from `::IUnknown`. This avoids accidentally creating a `com_ptr` from a C++/WinRT ABI pointer (which are in the `impl` namespace and are therefore off-limits).

If you feel so brave, you can also create a maker function that copies the ABI pointer.

```
template<typename T>
winrt::com_ptr<T>
    std::enable_if_t<std::<is_base_of_v<::IUnknown, T>, T>>
    make_com_ptr_from_copy(T* p)
{
    winrt::com_ptr<T> result;
    result.copy_from(p);
    return result;
}
```

Bonus chatter: The last time I made a survey of C++ COM wrappers, someone asked me to include C++/CX in the list. But C++/CX doesn't support wrappers around classic COM interfaces; it supports only Windows Runtime interfaces. Therefore, the answer for C++/CX is always "Not supported. Nothing to see here."