Summary of the duck-typing requirements of C++ COM wrappers

devblogs.microsoft.com/oldnewthing/20230516-00

May 16, 2023

Raymond Chen

Here's a summary of the tables we spent the past few days building. Points of interest are highlighted.

	`_com_ptr_t`	MFC `IPTR`	ATL `CComPtr`	WRL `ComPtr`
Default construction	Pass	Pass	Pass	Pass
Construct from raw pointer	Pass	Pass	Pass	Pass
Copy construction	Pass	Pass	Pass	Pass
Destruction	Pass	Pass	Pass	Pass
Attach and detach	Pass	Pass	Pass	Pass
Assign to same-type raw pointer	Pass	Pass	Pass	Pass
Assign to same-type smart pointer	Pass	Pass	Pass	Pass
Fetch the wrapped pointer	`GetInterfacePtr()`	`GetInterfacePtr()`	`p`, or implicit conversion	`Get()`

Access the wrapped object	`->`	`->`	`->` (suboptimal)	`->`
Receive pointer via `&`	release old	release old	must be empty	release old
Release and receive pointer	`&`	`&`	`&`	`ReleaseAndGetA`
Preserve and receive pointer	N/A	N/A	`&p`	`GetAddressOf()`
Return to empty state	Pass	Pass	Pass	Pass
Comparison	Fail	N/A	Pass	Pass
Accidental bypass	Fail	Fail	Pass	Fail
Construct from other-type raw pointer	Pass	Pass	Pass	Pass
Construct from other-type smart pointer	Pass	Pass	Pass	Pass
Assign from other-type raw pointer	Pass	Pass	Pass	Pass
Assign from other-type smart pointer	Pass	Pass	Pass	Pass
`T` can be `final`	Yes	Yes	No	Yes
`T::Release` must return `ULONG`	No	No	Yes	Yes
`T::Release` must be `__stdcall`	No	No	Yes	No

For the most part, all of the smart pointer wrappers can wrap pointers to non-COM objects, provided they have `AddRef` and `Release` methods which increment and decrement the object reference count.

ATL is the only one that protects against accidental bypass by using the "coloring" technique. However, the cost is that the wrapped object may not be `final`. Furthermore, its implementation of "coloring" is suboptimal if the underlying object does not derive from `IUnknown`. (If it does, then the "coloring" has no overhead.)

The semantics of the `&` address-of operator vary greatly, a topic I had discussed some time ago. ATL is the only one whose `&` operator does not release the pointer before producing the address. C++/WinRT does not overload the `&` operator at all.

`_com_ptr_t`, `IPTR`, and C++/WinRT do not provide a way to access the wrapped pointer's address without freeing it.

Comparison operators break only in `_com_ptr_t`. (Comparison operators never worked with `IPTR`, so there was nothing to break.)

C++/WinRT does not support constructing directly from a raw pointer. This is annoying because the case we are most likely to use it for a non-IUnknown-derived class is specifically to construct directly from `this`:

```
// Sadly doesn't work
RegisterCallback([strongThis = winrt::com_ptr(this)] { ... });
```

ATL, WRL, and wil require that the `Release` method return the new reference count as a `ULONG`. ATL even requires that the `Release` method be `__stdcall`.

The MFC `IPTR` macro is not really used any more, but I listed it in the table for completeness. If you try to use it in a modern compiler, you will get warnings about its use of exception specifiers, which were deprecated in C++17.

Glancing through the table, it seems that the least weird wrapper classes for wrapping types that aren't COM interfaces are `WRL::ComPtr` and `wil::com_ptr`. You do have to make sure that your `Release()` method returns the new reference count, but this is likely something you already have readily available, so returning it is not that big of an obstacle.