Once your object reaches final_release, you are committed to destructing it (eventually)

devblogs.microsoft.com/oldnewthing/20240221-00

February 21, 2024

Raymond Chen

Some time ago, I discussed C++/WinRT implementation extension points, specifically the `final_release` extension point which is given a `std::unique_ptr` of the object which just experienced its final release. Since it happens before the destructor is run, you have extra flexibility in how you run the object down. A common reason for using `final_release` is to ensure that the destructor runs in a specific execution environment. For example, you can use it to ensure that the object's destructor runs on a specific thread.

But one thing to keep in mind is that when you receive an object in your `final_release`, it is no longer a COM object. It is now just a C++ object that is on its way to destruction. The expectation is that you will eventually destruct the object, though you may perform various cleanup operations before finally getting around to the destructor.

Since it's no longer a COM object, you can't resurrect it and extend its COM lifetime. Its COM lifetime is over.

For example, any outstanding COM weak references to the object will no longer resolve. As far as those weak references are concerned, the object is already gone. Even if you extend its C++ lifetime, its COM lifetime is already over. It has disappeared from the world of COM.

The fact that the object's COM lifetime has ended means that you can't use `final_release` to solve the `IMemoryBufferReference.Closed` problem. If you to raise the `Closed` event from `final_release`, you're giving out a COM reference to something that is no longer a COM object. I mean, you could still try to use it like a COM object, but when your `unique_ptr` destructs, the object will destruct even if the `Closed` event handler performed its own `AddRef` to extend the object's lifetime.

In theory, `final_release` could have been designed so that it is called before the weak references are disconnected, and while the object is still a valid COM object whose lifetime can be extended. It's bit tricky because you can't just blindly subtract the strong reference; you have to prevent it from dropping to zero and ending the COM lifetime. But you don't want that if your `final_release` is for cleaning up the object, because that means that an outstanding COM weak reference could reacquire a strong reference to the COM object while you are in the process of cleaning it up. There would have to be a special function like `disconnect_weak_references` for `final_release` to call. Not only would people probably forget to call it, but in the common case where you never intended to resurrect the object, the C++/WinRT infrastructure went to a lot of effort to keep the weak references connected, only for you to immediately tell it, "Oh, nevermind."

Since the vast majority of usage of `final_release` is cleaning up the object in an organized manner, cases for which you want the weak references to be disconnected, C++/WinRT is biased toward that common case. If you want the other rare case, you can use the trick we used when we implemented the `IMemoryBufferReference.Closed` event.