Creating a non-agile delegate in C++/WinRT, part 1: Initial plunge

devblogs.microsoft.com/oldnewthing/20200406-00

April 6, 2020

Raymond Chen

Delegates in C++/WinRT are agile by default. This means that they can run on any thread. This is normally a good thing.

There is a weird corner case where you may want a non-agile delegate, however. It requires that various stars align in just the right (wrong) way.

First, the event source must apply specific semantics to the fact that the event handler has returned. For example, it may assume that when the event handler returns, certain actions have been performed, such as setting a response property on the event arguments.

Most of the time, when this scenario occurs, the event arguments offer a deferral. This lets the event handler say, "Okay, I'm not finished yet, so please wait for me to go do some more work. I'll let you know when I'm done." The reason why you'd need to do this is if some of your work involves coroutines:

```
object.SomethingHappened(
    [=](auto sender, auto e) -> winrt::fire_and_forget
    {
        auto lifetime = get_strong();
        auto deferral = e.GetDeferral();
        co_await resume_foreground(Dispatcher());
        e.Result(co_await calculate_result());
        deferral.Complete();
    });
```

(Note that we accept the parameters by value rather than by reference since we are a coroutine.)

But suppose your event source doesn't offer a deferral on its event arguments. I'm looking at you, DeviceWatcher.

Suppose further that your delegate needs to perform operations in a specific thread context. (If it didn't require a specific thread context, then the delegate can just do its work on whatever context it was invoked from.) The most common example of this is a delegate that manipulates user interface objects in response to the event.

Also assume that the event source raises the event in the "wrong" thread context, that is, a context different from the one you require. Most UI-related events are raised on the appropriate UI thread, so this is typically a problem only with events that are not UI events, but where you want to do UI work in the event handler.

And the last assumption for needing to use this trick is that the work you intend to perform on the specific thread context is all synchronous. (We'll look at this requirement some more later.)

If all the stars (mis-)align, a non-agile delegate may be useful. You can create one by using the `InvokeInContext` function I introduced back when we talked about COM contexts, and using the same technique we used when demonstrating how to use contexts to return to a COM apartment:

```
deviceWatcher.Added(
    [=, context = CaptureCurrentApartmentContext()]
    (auto&& sender, auto&& info)
    {
        InvokeInContext(context.Get(), [&]()
        {
          viewModel.Append(make<DeviceItem>(info));
        });
    });
```

Note that the parameters can be accepted by reference because the lambda runs synchronously.

Next time, we'll look more closely at this pattern.

Bonus chatter: I'm pulling a fast one here, because we didn't actually create a non-agile delegate. Instead, we created an agile delegate that behaves like a non-agile delegate. Which is basically just as good.

Raymond Chen

Follow