Creating an already-completed asynchronous activity in C++/WinRT, part 7

devblogs.microsoft.com/oldnewthing/20240717-00

July 17, 2024

Raymond Chen

So far, we've been building helpers for producing already-completed asynchronous activities by creating coroutines that immediately succeed or fail. But really, coroutines are a red herring. What we want is an object that implements an asynchronous interface which models an already-completed asynchronous activity. In other words, we don't have to be a coroutine. We just have to be an interface.

As before, we'll start with a special case to make sure we understand the problem, and then we'll generalize it.

```
struct completed_action :
    winrt::implements<completed_action,
        winrt::Windows::Foundation::IAsyncAction,
        winrt::Windows::Foundation::IAsyncInfo>
{
    using namespace winrt::Windows::Foundation;

    winrt::slim_mutex m_mutex;
    winrt::AsyncActionCompletedHandler m_completed;

    // IAsyncInfo
    auto Id() { return 1; }
    auto Status() { return AsyncStatus::Completed; }
    auto ErrorCode() { return S_OK; }
    auto Cancel() { }
    auto Close() { }

    // IAsyncAction
    auto Completed()
    {
        winrt::slim_lock_guard lock(m_mutex);
        return m_completed;
    }
    auto Completed(winrt::AsyncActionCompletedHandler const& handler)
    {
        {
            winrt::slim_lock_guard lock(m_mutex);
            if (m_completed) {
                throw winrt::hresult_illegal_delegate_assignment();
            }
            m_completed = handler;
        }
        handler(*this, Status());
    }
    auto GetResults() { }
};
```

This implements the `IAsyncAction` contract in a very simple way: It is always `Completed`, there is no error, you can't cancel or close it, it has no results, and it immediately invokes the `Completed` handler when set.

The most complicated part is the `Completed` handler because it's nearly all just bookkeeping. First, we enforce the rule that you can set the handler only once. And then we remember the handler so that we can return it if anybody ever asks for it. (Narrator: Nobody ever asks for it.) The only real work is invoking the handler immediately to tell it "I'm already done!"

We can use this already-completed action when have something that can complete synchronously, but the interface expects an asynchronous action.

```
winrt::IAsyncAction SetNameAsync(winrt::hstring const& name)
{
    m_name = name;
    return winrt::make<completed_action>();
}
```

For an already-completed `IAsyncOperation<T>`, everything would be the same, except that our `GetResults()` returns the already-completed value.

And if we are implementing a `-WithProgress`, then we also need a `Progress` delegate property that we never call, but nevertheless have to remember in case the caller reads the property back.

```
    winrt::AsyncActionProgressHandler<P> m_progress;

    auto Progress()
    {
        winrt::slim_lock_guard lock(m_mutex);
        return m_progress;
    }
    auto Progress(winrt::AsyncActionProgressHandler<P> const& handler)
    {
        winrt::slim_lock_guard lock(m_mutex);
        m_progress = handler;
    }
```

Next time, we'll write a generalized version, now that we understand the pattern.