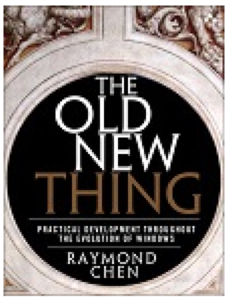
Adding delays to our task sequencer, part 1

devblogs.microsoft.com/oldnewthing/20250402-00/?p=111040

April 2, 2025



Suppose you want to use the `task_sequencer` class we created a while back (and which we fixed not too long ago), but you also want to implement a rudimentary form of throttling, so that tasks run at a specified maximum rate.

Suppose for concreteness that you want to have a 1-second cooling off period before the next task runs. How would we add this to our `task_sequencer`?

Well, the thing that kicks off the next task is the `completer`, which calls `complete()` on the chained task to trigger the start of the next task. All we have to do is delay that completion. For that, we can use `fire_and_forget`.

```
struct task_sequencer
{
    ⟦ ... ⟧

    struct completer
    {
        ~completer()
        {
            complete_later(std::move(chain));
        }
        std::shared_ptr<chained_task> chain;

        static fire_and_forget complete_later(
            std::shared_ptr<chained_task> chain)
        {
            co_await winrt::resume_after(1s);
            chain->complete();
        }
    };

    ⟦ ... ⟧
};
```

Instead of calling `chain->complete()` immediately from the destructor, we kick off a coroutine that calls it after waiting one second.

This coroutine is simple enough you might find it easier to inline it, so that all the logic is in one place.

```
struct task_sequencer
{
    ⟦ ... ⟧

    struct completer
    {
        ~completer()
        {
            [](auto chain) -> winrt::fire_and_forget {
                co_await winrt::resume_after(1s);
                chain->complete();
            }(std::move(chain));
        }
        std::shared_ptr<chained_task> chain;
    };

    ⟦ ... ⟧
};
```

Maybe instead of waiting one second between the completion of one operation and the start of the next, you want to wait one second between the *start* of one operation and the start of the next. We’ll look at that next time.