# Expressing fire-and-forget coroutines more explicitly, -or- How to turn any coroutine into fire-and-forget

devblogs.microsoft.com/oldnewthing/20190321-00

March 21, 2019

Raymond Chen

Last time, we looked at how to mark a coroutine as fire-and-forget, meaning that the caller does not get any information about when the coroutine completes. This is fine as far as it goes, but it may not be what you want.

Fire-and-forget-ness is frequently a property of the call site, not the function itself. A particular coroutine could be called with a continuation in one case, but as fire-and-forget in other cases. There should be a way to capture the desired behavior at the call site because it's the caller's choice whether they want to wait for the result or to proceed without it.

```
using winrt::Windows::Foundation;

IAsyncAction DoSomethingAsync()
{
  co_await blah();
  co_await blah();
  co_await blah();
}

// This caller cares about when the coroutine completes.
IAsyncAction DoSomethingAndThenSomethingElseAsync()
{
  co_await DoSomethingAsync();
  DoSomethingElse();
}

// This caller doesn't care
void StartDoingSomethingAndSomethingElse()
{
  // Don't co_await this; just let it go.
  DoSomethingAsync();

  // This runs while the DoSomethingAsync is still in progress.
  DoSomethingElse();
}
```

Calling `DoSomethingAsync` and throwing away the `IAsyncAction` is dangerous: If an unhandled exception occurs in the task, there is nobody around to observe it, and you're back to where you were with the overly forgetful `winrt::fire_and_forget`.

On the other hand, we don't want to write two versions of `DoSomethingAsync`, one which returns an `IAsyncAction` and another which returns a `winrt::fire_and_forget`. We should be able to convert any `IAsyncAction` into a `winrt::fire_and_forget`.

```
template<typename T>
winrt::fire_and_forget no_await(T t)
{
 co_await t;
}
```

Now you can declare at the call site that you don't care about the completion (aside from ensuring that it doesn't trigger any unhandled exceptions).

```
IAsyncAction DoSomethingAndThenSomethingElseAsync()
{
  co_await DoSomethingAsync();

  // This doesn't run until the DoSomethingAsync completes.
  DoSomethingElse();
}

// This caller doesn't care
void StartDoingSomethingAndSomethingElse()
{
  // This starts and we don't want for it to complete.
  no_await(DoSomethingAsync());

  // This runs while the DoSomethingAsync is still in progress.
  DoSomethingElse();
}
```

This helper is useful when employed in conjunction with `invoke_async_lambda`.

```
void OnClick()
{
 no_await(invoke_async_lambda([=]() -> IAsyncAction
 {
   ... do stuff, including co_await ...
 }));
}
```

The combination is useful enough that you might want a helper that does both.

```
template<typename T>
winrt::fire_and_forget no_await_lambda(T t)
{
 co_await t();
}
```

Recall that the subtlety of `invoke_ async_ lambda` is that it copies the lambda into its frame, so that its lifetime will extend until the coroutine completes. But `no_ await` already copies the lambda into its frame, so the make work of `invoke_ async_ lambda` is already taken care of! All that's left is to `co_ await` it into a `winrt:: fire_ and_ forget`.

Next time, we'll try to unify `no_ await` and `no_ await_ lambda`, mostly because I think the name `no_ await` is really cute and I don't want to give it up.

Raymond Chen

**Follow**