Cancelling a Windows Runtime asynchronous operation, part 1: C#

devblogs.microsoft.com/oldnewthing/20200701-00

July 1, 2020

Raymond Chen

The Windows Runtime has a lot of asynchronous operations, and the typical way of dealing with them is to schedule a continuation when the operation completes. Depending on the language, it might be `Promise.then` or `task.then()` or `await` or `co_await`. They all boil down to "Return from the function immediately, and resume execution when this asynchronous thing produces a value."

Windows Runtime asynchronous operations can be cancelled, and then things get interesting.

Invoking the `Cancel` method is, technically, merely a request and not a demand. Operations are advised to support cancellation, but it is not strictly required. An operation might choose to ignore your cancellation request outright. Or it could process the cancellation request by causing the operation to complete immediately with partial results.

But let's assume that the operation accepted your cancel request and indeed completed the operation with a status of `Canceled`.

What does your code see as the result of the cancellation?

Let's start with C#. In order to cancel the operation, you have to wrap it in a task, and then cancel the task.

```
var picker = new FileOpenPicker { FileTypeFilter = { ".txt" } };
var cts = new CancellationTokenSource();
cts.CancelAfter(TimeSpan.FromSeconds(3));

StorageFile file;
try {
    file = await picker.PickSingleFileAsync().AsTask(cts.Token);
} catch (TaskCanceledException) {
    file = null;
}

if (file != null) {
    DoSomething(file);
}
```

We cancel the file picker dialog after three seconds. This is done by taking the `IAsyncOperation` returned by `PickSingleFileAsync()`, convert it to a `Task` with `AsTask`, and associate it with a cancellation token source that has been configured to cancel after three seconds.

When this operation is canceled, you get a `TaskCanceledException`. This was the result of the Windows Runtime asynchronous operation completing with a status of `Canceled`. The C# projection then converts this to a `TaskCanceledException`.

This is the behavior that C# asynchronous code expects, so it's natural that the C# projection of Windows Runtime asynchronous operations into tasks behaves this way.

Next time, we'll look at C++/CX with PPL.

Bonus chatter: You can also cancel the task by talking directly to the `IAsyncOperation` instead of converting it into a C# Task.

```
async void CancelAfter(IAsyncInfo info, TimeSpan delay)
{
    await Task.delay(delay);
    info.Cancel();
}

var picker = new FileOpenPicker { FileTypeFilter = { ".txt" } };
StorageFile file;
try {
    var op = picker.PickSingleFileAsync();
    CancelAfter(op, TimeSpan.FromSeconds(3));
    file = await op;
} catch (TaskCanceledException) {
    file = null;
}

if (file != null) {
    DoSomething(file);
}
```

You could try to earn some style points by moving the `CancelAfter` code inline.

```
var picker = new FileOpenPicker { FileTypeFilter = { ".txt" } };
StorageFile file;
try {
    var op = picker.PickSingleFileAsync();
    ((Action)(async () => { await Task.Delay(TimeSpan.FromSeconds(3)); op.Cancel();
}))();
    file = await op;
} catch (TaskCanceledException) {
    file = null;
}
```

Or perhaps more usefully, let `CancelAfter` return the original asynchronous operation, so you can cancel it and await it at one go.

```
public static class Helpers
{
    static async void CancelAfterHelper(IAsyncInfo info, TimeSpan delay)
    {
        await Task.Delay(delay);
        info.Cancel();
    }

    static public IAsyncAction CancelAfter(this IAsyncAction action, TimeSpan delay)
    {
        CancelAfterHelper(action, delay);
        return action;
    }

    static public IAsyncOperation<T>
        CancelAfter<T>(this IAsyncOperation<T> op, TimeSpan delay)
    {
        CancelAfterHelper(op, delay);
        return op;
    }
}

var picker = new FileOpenPicker { FileTypeFilter = { ".txt" } };
StorageFile file;
try {
    file = await picker.PickSingleFileAsync().CancelAfter(TimeSpan.FromSeconds(3));
} catch (TaskCanceledException) {
    file = null;
}
```

Raymond Chen

Follow