Why does IAsyncAction or IAsyncOperation.GetResults() produce a E_ILLEGAL_METHOD_CALL exception?

devblogs.microsoft.com/oldnewthing/20230724-00

July 24, 2023

Raymond Chen

For expository purposes, let's look at the code we wrote some time ago which obtains network usage information.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Windows.Networking.Connectivity;

class Program
{
    static async Task DoIt()
    {
        var now = DateTime.Now;
        var states = new NetworkUsageStates
        { Roaming = TriStates.DoNotCare, Shared = TriStates.DoNotCare };

        var profiles = NetworkInformation.GetConnectionProfiles();
        foreach (var profile in profiles)
        {
            var usages = await profile.GetNetworkUsageAsync(
                now.AddDays(-1), now, DataUsageGranularity.PerDay,
                states);
            var usage = usages[0];
            if (usage.ConnectionDuration > TimeSpan.Zero)
            {
                Console.WriteLine(profile.ProfileName);
                Console.WriteLine($"BytesReceived = {usage.BytesReceived}");
                Console.WriteLine($"BytesSent = {usage.BytesSent}");
                Console.WriteLine($"ConnectionDuration =
{usage.ConnectionDuration}");
                Console.WriteLine($"------------------");
            }
        }

    }
    static void Main()
    {
        DoIt().GetAwaiter().GetResult();
    }
}
```

But instead of wrapping the calculations inside a `DoIt` helper method, why not just grab the results directly?

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Windows.Networking.Connectivity;

[STAThread]
static void Main()
{
    var now = DateTime.Now;
    var states = new NetworkUsageStates
    { Roaming = TriStates.DoNotCare, Shared = TriStates.DoNotCare };

    var profiles = NetworkInformation.GetConnectionProfiles();
    foreach (var profile in profiles)
    {
        var usages = /* await */ profile.GetNetworkUsageAsync(
            now.AddDays(-1), now, DataUsageGranularity.PerDay,
            states).GetResults();
        var usage = usages[0];
        if (usage.ConnectionDuration > TimeSpan.Zero)
        {
            Console.WriteLine(profile.ProfileName);
            Console.WriteLine($"BytesReceived = {usage.BytesReceived}");
            Console.WriteLine($"BytesSent = {usage.BytesSent}");
            Console.WriteLine($"ConnectionDuration = {usage.ConnectionDuration}");
            Console.WriteLine($"------------------");
        }
    }
}
```

Unfortunately, if you try this, you discover that the `GetResults()` call always fails with the exception `E_ILLEGAL_METHOD_CALL`. What's going on?

There is a difference between `TaskAwaiter.GetResult()` and `IAsyncAction.GetResult()` (and its buddies like `IAsyncOperation<T>.GetResult()`).

The `TaskAwaiter.GetResult()` method waits for the task to complete before producing the results. On the other hand, `IAsyncAction.GetResult()` does not wait. It gives you the result if the asynchronous activity has completed, or it throws `E_ILLEGAL_METHOD_CALL` if the activity has not yet run to completion. You are expected to wait for the `Completed` callback before retrieving the result.[1]

Now in this case, the situation is compounded by the fact that the program also marked the `Main` method as `[STAThread]`. Single-threaded apartments must pump messages while waiting; otherwise, you may run into deadlocks. So maybe it's a good thing that this doesn't work. Otherwise, you'd be tempted to use it in an incorrect way.

[1] You can find the code that enforces this in the method `CheckValidStateForResultsCall`. We looked at this method some time ago.