Obtaining network usage information from the Windows Runtime

devblogs.microsoft.com/oldnewthing/20210520-00

May 20, 2021

Raymond Chen

Network usage information on Windows can be obtained from classes in the `Windows.Networking.Connectivity` namespace. The `NetworkInformation` class is your starting point.

We'll start with C# and translate to C++/WinRT when we're done.

Prepare a C# console application to use Windows Runtime asynchronous operations as described last time, and start typing.

```
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();
    }
}
```

All of the work happens in the creatively-named `DoIt` method. The main function just calls `DoIt()` and waits for the task to complete. If you can upgrade to C# 7.1 or better, you can take advantage of async main support.

Okay, so back to the `DoIt()` method.

We first capture the current time into a local variable `now`. This ensures that our time queries are consistent through the loop.

We also create a local `NetworkUsageStates` object which says that we don’t care about distinguishing whether you were Roaming or Shared.

We then ask the `NetworkInformation` object for all the connection profiles. For each profile, we ask for usage in the last 24 hours, aggregated by day. Given those parameters, there will be exactly one usage report.[1]

For any connection that was used for a nonzero amount of time, we report the various properties of that daily usage.

Note that generating these reports is time-consuming, so if you already know which connections you want, you can filter on their name. If you are interested only in the connection that is currently being used for Internet access, you can use `GetInternetConnectionProfile` instead of enumerating through all of the connections.

And here's the C++/WinRT version.

```
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Foundation.Collections.h>
#include <winrt/Windows.Networking.Connectivity.h>
#include <stdio.h>

using namespace std::literals::chrono_literals;
using namespace winrt;
using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::Networking::Connectivity;

IAsyncAction DoIt()
{
    auto now = clock::now();
    NetworkUsageStates states{ TriStates::DoNotCare, TriStates::DoNotCare };

    auto profiles = NetworkInformation::GetConnectionProfiles();
    for (auto profile : profiles)
    {
        auto usages = co_await profile.GetNetworkUsageAsync(
            now - 24h, now, DataUsageGranularity::PerDay, states);
        auto usage = usages.GetAt(0);
        auto seconds = static_cast<int>(std::chrono::duration_cast<
            std::chrono::seconds>(usage.ConnectionDuration()).count());
        if (seconds > 0)
        {
            printf("%ls\n", profile.ProfileName().c_str());
            printf("BytesReceived = %I64u\n", usage.BytesReceived());
            printf("BytesSent = %I64u\n", usage.BytesSent());
            printf("ConnectionDuration = %d seconds\n", seconds);
            printf("-----------------\n");
        }

    }
}

int main()
{
    init_apartment();
    DoIt().get();
}
```

Next time, we'll look at usage attribution.

[1] If we had asked for multiple days, then there would be one report per day, in chronological order. Note that the definition of "day" is not "calendar day" but "24-hour period starting at the provided start time." If the time range is not an exact multiple of the granularity, then the last report will cover only part of the granularity interval.

Raymond Chen

Follow