How can I ask the networking stack whether the system has obtained network connectivity?

devblogs.microsoft.com/oldnewthing/20201102-00

November 2, 2020

Raymond Chen

Recall a while back we had a customer who had assumed that the `GetIpAddrTable` function returned the IP addresses sorted by scope rather than by IP address. Digging into the customer's actual problem revealed that what they really wanted to know was when the system had obtained network connectivity.

Their original plan was to poll on `GetIpAddrTable` expecting their public IP address to show up first, but we learned that that technique was flawed. So what can they do? They wondered if maybe there was a specific service whose startup demonstrated that the system had an IP address.

Network connectivity is more fickle than just checking for a service. It can come and go dynamically. You wander in and out of range of a Wi-Fi base station. You plug in or remove the USB networking adapter. The cleaning crew temporarily—or perhaps permanently—unplugs the Ethernet cable because it prevents them from vacuuming the floor.

The networking folks had a few suggestions. One was to use `INetworkList-Manager:: get_ IsConnected`, which tells you if the system has any network connectivity at all. This will tell you whether the system is connected to the local network, if that's what you care about.

If you are looking for internet connectivity, then you can use `INetworkList-Manager:: get_ IsConnectedToInternet`.

The methods on `INetworkListManager` aggregates network connectivity across multiple interfaces, saving you the trouble of having to deal with the messy details of the networking stack.

If you prefer to get your hands dirty, you can use the `NotifyUnicastIpAddressChange` function which lets you register a callback that will be invoked whenever a change occurs to the IP address of the system (optionally filtered to only IPv4 or IPv6).

This function is a little tricky because your callback doesn't actually get the address information. Instead, it contains only enough information for you to call `GetUnicastIp-AddressEntry` to obtain more information about the new address. In particular, you want to see if the IP address meets your criteria (routability being one thing you probably want) and that the Duplicate Address Detection (DAD) state is *Preferred*, indicating that this is the preferred IP address.

Another high-level interface is the network connection profile in the Windows Runtime.

```
using Windows.Networking.Connectivity;

var level = NetworkConnectivityLevel.None;
var profile = NetworkInformation.GetInternetConnectionProfile();
if (profile != null) {
  level = profile.GetNetworkConnectivityLevel();
}
```

The `GetInternetConnectionProfile` method gives you a `ConnectionProfile` object which describes the system's internet connection. This also tries to distinguish between normal internet connectivity and being trapped in a captive portal. You can subscribe to the `NetworkStatusChanged` event to be notified when the connectivity level changes. (The Windows Runtime class also gives you other information, such as whether you are on a metered network.)

You can see from the extra features in the Windows Runtime class that it is more focused on app scenarios (particularly mobile networking) than on understanding the low-level details of the networking stack. This particular customer was writing software that ran in a server room, so they didn't need to worry about things like metered networks or captive portals.

Or at least I hope they didn't!

Raymond Chen

Follow