Why does the wireless connection dialog ask for your password twice?

devblogs.microsoft.com/oldnewthing/20100419-00

April 19, 2010

Raymond Chen

Martin wonders why the wireless networking dialog asks you to type your password twice when connecting to an existing network.

Yeah, that bothers me too, and I don't know why either.

But while we're on the topic of wireless networking, I thought I'd share a little program that is just as useless as my answer above. (If other people get to hijack the topic, then I want to also.)

Back in the early days of Windows XP, I found that my wireless networking adapter would constantly disconnect and reconnect. I never figured out why, but I did have a theory. (Theory: The wireless zero configuration service saw another access point and said, "Hey, that access point over there looks much nicer than then one I'm currently connected to. I'm going to drop my current connection and see if maybe that other access point will go out with me." And then it went up to that other access point and asked it out on a date. When the other access point said no, it came crawling back to the original access point. Repeat.)

Anyway, to avoid this problem (which went away after a while for reasons unclear; maybe it was fixed, maybe whatever situation triggered the problem went away, I didn't bother investigating), I wrote a program which did two very simple things:

1. If the wireless networking adapter was connected to an access point, then turn off the wireless zero configuration service.
2. If the wireless networking adapter was not connected to an access point, then turn on the wireless zero configuration service.

In other words, it automates the process described on this Web page. (I like how that article was copied in its entirety to another site, which replaced the author's name. Now that's chutzpah.)

Mind you, the program really is no longer interesting in and of itself any more because the underlying problem went away, but I thought it could serve as an illustration of how you can put together some simple things to make a useful tool.

First, I changed the security descriptor on the wireless zero configuration service so that my account had permission to turn it on and off.

Second, I added this code to a program that hangs out my Startup group which monitors various things I like to monitor. (I have one program that monitors several things just to cut down on the number of processes hanging around on my machine.) The code has been compressed and reformatted to get rid of the uninteresting parts.

```
class MonitorWireless
{
public:
  MonitorWireless()
    : m_hWait(NULL)
  {
      ZeroMemory(&m_o, sizeof(m_o));
  }
  ~MonitorWireless()
  {
    if (m_hWait) UnregisterWaitEx(m_hWait, INVALID_HANDLE_VALUE);
    if (m_o.hEvent) CloseHandle(m_o.hEvent);
  }
  BOOL Initialize();
protected:
  static void CALLBACK s_OnChange(PVOID lpParameter, BOOLEAN)
  {
    MonitorWireless *self =
               reinterpret_cast<MonitorWireless*>(lpParameter);
    self->CheckIPAddress(); // something changed - check it again
  }
  void CheckIPAddress();
  static void StartStopService(BOOL fStart);
private:
    HANDLE m_hWait;
    OVERLAPPED m_o;
}
```

The class definition is all very boring. Our class has an `OVERLAPPED` structure which we use to register for IP address change notifications, and it has a handle to a registered wait, which takes advantage of the thread pool to reduce the number of threads used by the process.

```
BOOL MonitorWireless::Initialize()
{
  m_o.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
  if (!m_o.hEvent) return FALSE;
  if (!RegisterWaitForSingleObject(&m_hWait, m_o.hEvent,
                      s_OnChange, this, INFINITE, 0)) return FALSE;
  CheckIPAddress();
  return TRUE;
}
```

When the object is initialized, it creates the handle that we will ask to be set whenever the computer's IP address changes, and then registers a wait on that handle with a callback function. When the event is signaled, we check the IP address. And to start the ball rolling, we check the IP address at initialization.

```
void MonitorWireless::CheckIPAddress()
{
  ULONG ulSize = 0;
  if (GetIpAddrTable(NULL, &ulSize, 0) ==
                                     ERROR_INSUFFICIENT_BUFFER) {
    PMIB_IPADDRTABLE piat = reinterpret_cast<PMIB_IPADDRTABLE>
                               (LocalAlloc(LMEM_FIXED, ulSize));
    if (piat) {
      if (GetIpAddrTable(piat, &ulSize, 0) == ERROR_SUCCESS) {
        BOOL fFound = FALSE;
        for (DWORD dwIndex = 0; dwIndex < piat->dwNumEntries;
             dwIndex++) {
          PMIB_IPADDRROW prow = &piat->table[dwIndex];
          if (prow->dwAddr == 0) continue;
          if ((prow->wType & (MIB_IPADDR_DYNAMIC |
                              MIB_IPADDR_DELETED |
                              MIB_IPADDR_DISCONNECTED)) !=
                              MIB_IPADDR_DYNAMIC) continue;
          fFound = TRUE;
          break;
        }
        StartStopService(!fFound);
      }
      LocalFree(piat);
    }
  }
  HANDLE h;
  NotifyAddrChange(&h, &m_o);
}
```

We start by getting the IP address table (doing the standard two-step of first asking how much memory we need to hold it, allocating the memory, and then filling the buffer) and walking through each IP address. If we find an entry with an IP address that is dynamic, not deleted, and not disconnected, then we declare ourselves happy; otherwise we are sad. If we are happy, then we stop the wireless zero configuration service; if we are sad, then we start it.

```
void MonitorWireless::StartStopService(BOOL fStart)
{
  SC_HANDLE sc;
  sc = OpenSCManager(NULL, NULL, SC_MANAGER_CONNECT |
                                 SC_MANAGER_ENUMERATE_SERVICE);
  if (sc) {
    SC_HANDLE scWzcsvc = OpenService(sc, TEXT("wzcsvc"),
                  fStart ? SERVICE_START
                         : SERVICE_STOP | SERVICE_QUERY_STATUS);
    if (scWzcsvc) {
      if (fStart) StartService(scWzcsvc, 0, NULL);
      else        StopService(scWzcsvc);
      CloseServiceHandle(scWzcsvc);
    }
    CloseServiceHandle(sc);
  }
}
```

To start or stop the service, we first connect to the service control manager, open the service we want to start/stop, and then, well, start or stop it.

There is already a `StartService` function, but no `StopService` function, so I wrote my own:

```
void StopService(SC_HANDLE sc)
{
 SERVICE_STATUS ss;
 if (QueryServiceStatus(sc, &ss) &&
     ss.dwCurrentState != SERVICE_STOPPED &&
     ss.dwCurrentState != SERVICE_STOP_PENDING)
   ControlService(sc, SERVICE_CONTROL_STOP, &ss);
}
```

If the service is not already stopped (or stopping), then we tell it to stop.

And there you have it, a program that you don't need any more. But the point here was more to show how you can put together some basic elements to solve a simple problem.

Techniques illustrated:

- Registering a wait in the thread pool.
- Registering asynchronously for IP address changes.
- Starting and stopping a service.

Raymond Chen

Follow