How do I enumerate remembered connections that are not currently connected?

devblogs.microsoft.com/oldnewthing/20150831-00

August 31, 2015

Raymond Chen

Harry Johnston wanted to know how to get a list of remembered (but not currently connected) drive mappings.

The idea here is to make a tweak to the Little Program. Start with what we had and make these changes:

```
int __cdecl main(int, char **)
{
 HANDLE hEnum;
 WNetOpenEnum(RESOURCE_REMEMBERED,
              RESOURCETYPE_DISK,
              0,
              NULL,
              &hEnum);

 ...
}
```

This changes the program from enumerating connected resources to enumerating remembered resources.

The last step is to skip the remembered resources that are also connected. But this part is not Win32 programming; it's just programming, For each remembered resource, check if the `lpLocalName` is non-null and matches an `lpLocalName` that came out of an enumeration of connected resources.

So let's do it. We start with the header files:

```
#define UNICODE
#define _UNICODE
#define STRICT
#include <windows.h>
#include <stdio.h> // horrors! Mixing C and C++ I/O!
#include <string>
#include <set>
#include <memory>
#include <winnetwk.h>
```

Since we are using classes like `std::set` which throw exceptions, we need to wrap our resources inside RAII classes. Here's one for network resource enumeration:

```
class CNetEnumerator
{
public:
 CNetEnumerator() = default;
 ~CNetEnumerator() { if (m_hEnum) WNetCloseEnum(m_hEnum); }
 operator HANDLE() { return m_hEnum; }
 HANDLE* operator&() { return &m_hEnum; }
private:
 HANDLE m_hEnum = nullptr;
};
```

Here is our function to enumerate all network resources. It uses a callback because arghhhhhhhhhh wishes it were so.

```
template<typename Callback>
void for_each_network_resource(
    DWORD dwScope,
    DWORD dwType,
    DWORD dwUsage,
    LPNETRESOURCE pnrIn,
    Callback callback)
{
 CNetEnumerator hEnum;
 WNetOpenEnum(dwScope, dwType, dwUsage, pnrIn, &hEnum);

 const DWORD elements = 65536 / sizeof(NETRESOURCE);
 static_assert(elements > 1, "Must have room for data");
 std::unique_ptr<NETRESOURCE> buffer(new NETRESOURCE[elements]);

 DWORD err;
 do {
  DWORD cEntries = INFINITE;
  DWORD cb = elements * sizeof(NETRESOURCE);
  err = WNetEnumResource(hEnum, &cEntries, buffer.get(), &cb);
  if (err == NO_ERROR || err == ERROR_MORE_DATA) {
   for (DWORD i = 0; i < cEntries; i++) {
    callback(&buffer[i]);
   }
  }
 } while (err == ERROR_MORE_DATA);
}
```

There is a bit of trickery to get the enumeration buffer into a form that C++ likes. We had previously used `LocalAlloc`, which is guaranteed to return memory suitably aligned for `NETRESOURCE`. However, we can't do it for `new BYTE[]`, since that returns only byte-aligned data. We solve this problem by explicitly allocating `NETRESOURCE` objects, but choosing a number so that the result is close to our desired buffer size.[1]

We need another helper class so we can create a case-insensitive set.

```
struct CaseInsensitiveWstring
{
 bool operator()(const std::wstring& a, const std::wstring& b) const {
  return CompareStringOrdinal(a.c_str(), a.length(),
                              b.c_str(), b.length(), TRUE) == CSTR_LESS_THAN;
 }
};
```

Okay, now we can start doing actual work:

```
void report(PCWSTR pszLabel, PCWSTR pszValue)
{
 printf("%ls = %ls\n", pszLabel, pszValue ? pszValue : L"(null)");
}

int __cdecl wmain(int, wchar_t **)
{
 std::set<std::wstring, CaseInsensitiveWstring> connected;

 // Collect the local resources which are already connected.
 for_each_network_resource(RESOURCE_CONNECTED,
  RESOURCETYPE_DISK, 0, nullptr, [&](LPNETRESOURCE pnr) {
   if (pnr->lpLocalName != nullptr) {
    connected.emplace(pnr->lpLocalName);
   }
  });

 // Now look for remembered resources that are not connected.
 for_each_network_resource(RESOURCE_REMEMBERED,
  RESOURCETYPE_DISK, 0, nullptr, [&](LPNETRESOURCE pnr) {
   if (pnr->lpLocalName == nullptr ||
       connected.find(pnr->lpLocalName) == connected.end()) {
    report(L"localName", pnr->lpLocalName);
    report(L"remoteName", pnr->lpRemoteName);
    report(L"provider", pnr->lpProvider);
    printf("\n");
   }
  });

 return 0;
}
```

Not exciting. Mostly consists of boring typing. But hey, that's what programming is like most of the time.

[1] If we were being super-weenies about the buffer size, we could have written

```
 union EnumBuffer {
  BYTE bytes[65536];
  NETRESOURCE nr;
 };

 std::unique_ptr<EnumBuffer> buffer(new EnumBuffer());
 LPNETRESOURCE pnr = &buffer->nr;
 ...
  DWORD cb = sizeof(EnumBuffer);
```

Raymond Chen

Follow