Reading a contract from the other side: Application publishers

devblogs.microsoft.com/oldnewthing/20040831-00

August 31, 2004

Raymond Chen

In an earlier article, I gave an example of reading a contract from the other side. Here's another example of how you can read a specification and play the role of the operating system.

I chose this particular example because somebody wanted to do this and didn't realize that everything they needed was already documented; they just needed to look at the documentation in a different light.

The goal today is to mimic the list of programs that appears on the "Add New Programs" page of the Add or Remove Programs control panel. Note that in order for this list to appear at all, you need to be on a domain, and for the list to be nonempty, your domain controller needs to publish applications for domain members to install. Consequently, I suspect many of my readers won't get to see any interesting results from this exercise, but then again, the point of exercise is not the result but merely the doing of it.

The documentation for the IAppPublisher interface describes how a publisher can register itself to appear in the list of programs that can be installed. All you have to do is read the documentation from the other side: Instead of reading documentation about a method and asking, "How would I implement this method?", read it and ask, "How would I call this method?"

The documentation says that an app publisher registers its CLSID under the specified registry key. Therefore, if you want to find all the app publishers, you need to enumerate that key.

```
#define REGSTR_PATH_PUBLISHERS  \
    L"Software\\Microsoft"      \
    L"\\Windows\\CurrentVersion\\App Management\\Publishers"

    HKEY hkPub;
    if (RegOpenKeyExW(HKEY_LOCAL_MACHINE, REGSTR_PATH_PUBLISHERS,
                      0, KEY_READ, &hkPub) == ERROR_SUCCESS) {
      WCHAR szKey[MAX_PATH];
      for (DWORD dwIndex = 0;
           RegEnumKeyW(hkPub, dwIndex, szKey, MAX_PATH) == ERROR_SUCCESS;
           dwIndex++) {
        ...
      }
      RegCloseKey(hkPub);
    }
```

The documentation says that the subkeys have the CLSID in REG_SZ format, so that's what we read out.

```
        WCHAR szCLSID[MAX_PATH];
        LONG l = sizeof(szCLSID) - sizeof(WCHAR);
        if (RegQueryValueW(hkPub, szKey, szCLSID, &l) == ERROR_SUCCESS)
          szCLSID[l/sizeof(WCHAR)] = 0;
          CLSID clsid;
          if (SUCCEEDED(CLSIDFromString(szCLSID, &clsid))) {
            ...
          }
        }
```

Notice the extra care we take to avoid the problem of registry strings that aren't null-terminated, as discussed in an earlier entry.

The documentation quite explicitly states how this CLSID is used.

> **Add/Remove Programs** creates an instance of your object by calling CoCreateInstance for your object and requests the approprite [sic] **IAppPublisher** interface when the **Add New Programs** view is populated.

Not much choice, now, is there. So we do what it says.

```
            IAppPublisher *ppub;
            if (SUCCEEDED(CoCreateInstance(clsid, NULL,
                          CLSCTX_ALL, IID_IAppPublisher,
                         (void**)&ppub))) {
              ...
              ppub->Release();
            }
```

Okay, now that we have an app publisher, we can invoke the various methods on it to get information from that publisher. If we were more ambitious, we could ask for the categories but today we're just going to be happy with enumerating the programs so we can print their names.

```
IEnumPublishedApps *penum;
if (SUCCEEDED(ppub->EnumApps(NULL, &penum))) {
  IPublishedApp *papp;
  while (penum->Next(&papp) == S_OK) {
    ...
    papp->Release();
  }
  penum->Release();
}
```

The enumerator gives us an application interface, and we can use that interface to get information about the application and print it out.

```
    APPINFODATA info = { sizeof(info) };
    info.dwMask = AIM_DISPLAYNAME;
    if (SUCCEEDED(papp->GetAppInfo(&info)) &&
        (info.dwMask & AIM_DISPLAYNAME)) {
      wprintf(L"%ls\n", info.pszDisplayName);
      CoTaskMemFree(info.pszDisplayName);
    }
```

We ask only for the display name, since that's all we're interested in today. In a more complicated program, we may ask for other data and would probably not release the IPublishedApp interface immediately, but rather hang onto it so we could invoke some other more interesting method like IPublishedApp::Install.

(Note that we have to use the correct memory allocator to free the memory.)

Okay, let's assemble all this into a simple console program.

```
#include <windows.h>
#include <ole2.h>
#include <shappmgr.h>
#include <stdio.h>

#define REGSTR_PATH_PUBLISHERS  \
    L"Software\\Microsoft"      \
    L"\\Windows\\CurrentVersion\\App Management\\Publishers"

int __cdecl main(int argc, char **argv)
{
  if (SUCCEEDED(CoInitialize(NULL)) {
    HKEY hkPub;
    if (RegOpenKeyExW(HKEY_LOCAL_MACHINE, REGSTR_PATH_PUBLISHERS,
                      0, KEY_READ, &hkPub) == ERROR_SUCCESS) {
      WCHAR szKey[MAX_PATH];
      for (DWORD dwIndex = 0;
           RegEnumKeyW(hkPub, dwIndex, szKey, MAX_PATH) == ERROR_SUCCESS;
           dwIndex++) {
        WCHAR szCLSID[MAX_PATH];
        LONG l = sizeof(szCLSID) - sizeof(WCHAR);
        if (RegQueryValueW(hkPub, szKey, szCLSID, &l) == ERROR_SUCCESS)
          szCLSID[l/sizeof(WCHAR)] = 0;
          CLSID clsid;
          if (SUCCEEDED(CLSIDFromString(szCLSID, &clsid))) {
            IAppPublisher *ppub;
            if (SUCCEEDED(CoCreateInstance(clsid, NULL,
                          CLSCTX_ALL, IID_IAppPublisher,
                         (void**)&ppub))) {
              IEnumPublishedApps *penum;
              if (SUCCEEDED(ppub->EnumApps(NULL, &penum))) {
                IPublishedApp *papp;
                while (penum->Next(&papp) == S_OK) {
                  APPINFODATA info = { sizeof(info) };
                  info.dwMask = AIM_DISPLAYNAME;
                  if (SUCCEEDED(papp->GetAppInfo(&info)) &&
                      (info.dwMask & AIM_DISPLAYNAME)) {
                    wprintf(L"%ls\n", info.pszDisplayName);
                    CoTaskMemFree(info.pszDisplayName);
                  }
                  papp->Release();
                }
                penum->Release();
              }
              ppub->Release();
            }
          }
        }
      }
      RegCloseKey(hkPub);
    }
    CoUninitialize();
```

```
  }
  return 0;
}
```

When you run this program, a list of all programs published by your domain controller should go scrolling past. (As I noted at the beginning of this entry, you won't see much if your computer is not on a domain or if your domain controller doesn't publish any programs.)

Yes, this program is not very pretty, because prettiness was not my goal. In real life, a lot of the mess would be moved out into helper functions, and you can clean it up even more by using a smart pointer library, but the goal here was not to write a pretty program; it was to show how something could be done by reading the specification from the other side.

(Why don't I use a smart pointer library? Because I try to write in "raw" C++ in order to avoid arguments about whose smart pointer library is best, or why smart pointers are evil… It's easy to convert "raw" C++ to use a smart pointer library, but it's harder to convert from one smart pointer library to another.)

Raymond Chen

Follow