A big little program: Monitoring Internet Explorer and Explorer windows, part 1: Enumeration

devblogs.microsoft.com/oldnewthing/20130610-00

June 10, 2013

Raymond Chen

Normally, Monday is the day for Little Programs, but this time I'm going to spend a few days on a single Little Program. Now, this might very well disqualify it from the name *Little Program*, but the concepts are still little; all I'm doing is snapping blocks together. (Plus, it's my Web site, so you can just suck it.)

The goal of our Little Program is to monitor Internet Explorer and Explorer windows as they are created, navigate to new locations, and are destroyed. (In principle, other Web browsers can participate in this protocol, but I don't know of any that do, so I'll assume that only Explorer and Internet Explorer register with the `ShellWindows` object.)

The key to all this is the `ShellWindows` object, which we've spent time playing with in the past.

Today we're going to write a helper function that takes an object returned by the `ShellWindows` object and extract the window handle and current location. This is the guts of our Little Program, so I'm basically doing the cool stuff up front, and then leaving the annoying bits for later.

```
#define UNICODE
#define _UNICODE
#define STRICT
#define STRICT_TYPED_ITEMIDS
#include <windows.h>
#include <ole2.h>
#include <iostream>


#include <shlobj.h>
#include <atlbase.h>
#include <atlalloc.h>
```

Now that we got the preliminary header file goop out of the way, we can write the exciting function.

```
HRESULT GetBrowserInfo(IUnknown *punk, HWND *phwnd,
                       PWSTR *ppszLocation)
{
 HRESULT hr;


 CComPtr<IShellBrowser> spsb;
 hr = IUnknown_QueryService(punk, SID_STopLevelBrowser,
                            IID_PPV_ARGS(&spsb));
 if (FAILED(hr)) return hr;


 hr = spsb->GetWindow(phwnd);
 if (FAILED(hr)) return hr;


 hr = GetLocationFromView(spsb, ppszLocation);
 if (SUCCEEDED(hr)) return hr;


 return GetLocationFromBrowser(punk, ppszLocation);
}
```

Awfully short for what purported to be an exciting function, but that's because I hid the exciting parts in helper functions.

First, we take the object and ask to locate the top-level browser, since that's where some of the interesting information hangs out. We ask for the `IShellBrowser` so we can get the window handle via the base class method `IOleWindow::GetWindow`. That's the easy part.

Getting the current location is tricky, because Explorer windows do it one way, and Internet Explorer does it another way. That's because Explorer windows browse the shell namespace, whereas Internet Explorer windows browse the Internet. Shell namespace locations are represented by pidls, whereas Internet locations are represented by URLs.

First, the Explorer way:

```
HRESULT GetLocationFromView(IShellBrowser *psb,
                            PWSTR *ppszLocation)
{
 HRESULT hr;


 *ppszLocation = nullptr;


 CComPtr<IShellView> spsv;
 hr = psb->QueryActiveShellView(&spsv);
 if (FAILED(hr)) return hr;


 CComQIPtr<IPersistIDList> sppidl(spsv);
 if (!sppidl) return E_FAIL;


 CComHeapPtr<ITEMIDLIST_ABSOLUTE> spidl;
 hr = sppidl->GetIDList(&spidl);
 if (FAILED(hr)) return hr;


 CComPtr<IShellItem> spsi;
 hr = SHCreateItemFromIDList(spidl, IID_PPV_ARGS(&spsi));
 if (FAILED(hr)) return hr;


 hr = spsi->GetDisplayName(SIGDN_DESKTOPABSOLUTEPARSING,
                           ppszLocation);
 return hr;
}
```

The maze we navigate here is to start from the `IShellBrowser` and get to the `IShellView` by calling `IShellBrowser::QueryActiveShellView`. It's rather annoying that the `IShellBrowser::QueryActiveShellView` method always returns you an `IShellView` rather than being forward-looking and letting you pass a `riid` / `ppv` pair. (The shell has for the most part learned this lesson, and new object creation or retrieval functions tend to take the `riid` / `ppv` pair so you can specify your ring size when you place the order instead of always getting a size 6 ring and then having to resize it.) Since `IShellBrowser::QueryActiveShellView` doesn't let us specify the desired interface, we have to do the `QueryInterface` ourselves to convert the `IShellView` into what we really want: The `IPersistIDList`.

From the `IPersistIDList` we ask for the pidl, which now tells us what the Explorer window is looking at. For display purposes, we convert it into a string by converting the pidl into an `IShellItem` (notice the handy `riid` / `ppv` pair produced by the type-checking `IID_PPV_ARGS` macro) and then asking the shell item for its parsing name.

(We saw techniques similar to this a few years ago.)

If it turns out we don't have an Explorer window, then we try again using the Web browser interfaces:

```
HRESULT GetLocationFromBrowser(IUnknown *punk,
                               PWSTR *ppszLocation)
{
 HRESULT hr;


 CComQIPtr<IWebBrowser2> spwb2(punk);
 if (!spwb2) return E_FAIL;


 CComBSTR sbsLocation;
 hr = spwb2->get_LocationURL(&sbsLocation);
 if (FAILED(hr)) return hr;


 return SHStrDupW(sbsLocation, ppszLocation);
}
```

We turn the object into an `IWebBrowser2` and ask for the `LocationURL` property. The annoyance here is that `IWebBrowser2` is an automation interface, so it uses `BSTR` for passing strings around, which is different from `IShellItem::GetDisplayName` which uses `CoTaskMemAlloc`, since that is the convention for non-dispatch COM interfaces. We therefore have to convert the `BSTR` to a task-allocated `PWSTR` before returning, so that the return value is consistent with `GetLocationFromView`.

Finally, we call the function in a loop to test that it actually works:

```
CComPtr<IShellWindows> g_spWindows;


HRESULT DumpWindows()
{
 CComPtr<IUnknown> spunkEnum;
 HRESULT hr = g_spWindows->_NewEnum(&spunkEnum);
 if (FAILED(hr)) return hr;


 CComQIPtr<IEnumVARIANT> spev(spunkEnum);
 for (CComVariant svar;
      spev->Next(1, &svar, nullptr) == S_OK;
      svar.Clear()) {
  if (svar.vt != VT_DISPATCH) continue;


  HWND hwnd;
  CComHeapPtr<WCHAR> spszLocation;
  if (FAILED(GetBrowserInfo(svar.pdispVal, &hwnd,
                            &spszLocation))) continue;


   std::wcout << hwnd
              << L" "
              << static_cast<PCWSTR>(spszLocation)
              << std::endl;
 }
 return S_OK;
}


int __cdecl wmain(int, PWSTR argv[])
{
 CCoInitialize init;
 g_spWindows.CoCreateInstance(CLSID_ShellWindows);
 DumpWindows();
 g_spWindows.Release();


 return 0;
}
```

Yes, I stupidly made `g_spWindows` a global variable, but it'll come in handy later. (It's still stupid, but at least there's a reason for the stupidity.)

Okay, we can take this program for a spin. When you run it, it should print the window handles and locations of all your Explorer and Internet Explorer windows.

Before we can start hooking up events to keep this list up to date, we need to learn a bit about connection points and using dispatch interfaces as connection point interfaces. We'll spend a few days on those topics, then return to our program.

Raymond Chen

Follow