A big little program: Monitoring Internet Explorer and Explorer windows, part 3: Tracking creation and destruction

devblogs.microsoft.com/oldnewthing/20130614-00

June 14, 2013

Raymond Chen

Last time, we listener for window navigations. Today we'll learn about tracking window creation and destruction.

The events to listen to are the `DShellWindowsEvents`. The `WindowRegistered` event fires when a new window is created, and the `WindowRevoked` event fires when a window is destroyed.

The bad news is that the parameter to those events is a cookie, which is not useful for much, so we just use the events to tell us that it's time to kick off a new enumeration to see what changed. This will also catch the case where something fell out of sync because a window closed without unregistering (say, because the application crashed).

Take our program from last time and make these changes:

```
LONG g_lCounter;


struct ItemInfo
{
 ItemInfo(HWND hwnd, IDispatch *pdisp)
  : hwnd(hwnd), lCounter(g_lCounter) { … }
 …


 HWND hwnd;
 CComPtr<CWebBrowserEventsSink> spSink;
 LONG lCounter;
};
```

The counter is used to detect stale windows when we re-enumerate.

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


 ++g_lCounter;


 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;


  ItemInfo *pii = GetItemByWindow(hwnd, nullptr);
  if (pii) { pii->lCounter = g_lCounter; continue; }
  pii = new(std::nothrow) ItemInfo(hwnd, svar.pdispVal);
  if (!pii) continue;


  LVITEM item;
  item.mask = LVIF_TEXT | LVIF_PARAM;
  item.iItem = MAXLONG;
  item.iSubItem = 0;
  item.pszText = spszLocation;
  item.lParam = reinterpret_cast<LPARAM>(pii);
  int iItem = ListView_InsertItem(g_hwndChild, &item);
  if (iItem < 0) delete pii;
 }


 int iItem = ListView_GetItemCount(g_hwndChild);
 while (–iItem >= 0) {
  ItemInfo *pii = GetItemByIndex(iItem);
  if (pii->lCounter != g_lCounter) {
   ListView_DeleteItem(g_hwndChild, iItem);
  }
 }


 return S_OK;
}
```

Building the window list is now a two-step process, since what we are really doing is *updating* the window list. First, we enumerate the contents of the `IShellWindows`. For each window, we get its window handle and see if there is already an item for that window. If so, then we update the counter for that item. If there is not already an item for that window, then we create one like we did before.

After we've processed all the windows that exist, we go look for the deletion by walking through all our items and deleting any whose counter was not updated by the previous loop.

Okay, but so far we haven't actually done anything new. Here's the new stuff:

```
class CShellWindowsEventsSink :
    public CDispInterfaceBase<DShellWindowsEvents>
{
public:
 HRESULT SimpleInvoke(
    DISPID dispid, DISPPARAMS *pdispparams, VARIANT *pvarResult)
 {
  switch (dispid) {
  case DISPID_WINDOWREGISTERED:
  case DISPID_WINDOWREVOKED:
   BuildWindowList();
   break;
  }
  return S_OK;
 }
};


CComPtr<CShellWindowsEventsSink> g_spShellSink;
```

This is the object that listens for changes to the window list. And whether the change is that a window arrived or a window departed, the response is the same: Refresh the window list.

All that's left to do is hook up this event sink (and clean it up):

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 g_hwndChild = CreateWindow(WC_LISTVIEW, 0,
    LVS_LIST | WS_CHILD | WS_VISIBLE |
    WS_HSCROLL | WS_VSCROLL, 0, 0, 0, 0,
    hwnd, (HMENU)1, g_hinst, 0);
 g_spWindows.CoCreateInstance(CLSID_ShellWindows);
 BuildWindowList();


 g_spShellSink.Attach(new CShellWindowsEventsSink());
 g_spShellSink->Connect(g_spWindows);


 return TRUE;
}


void OnDestroy(HWND hwnd)
{
 g_spWindows.Release();
 if (g_spShellSink) {
  g_spShellSink->Disconnect();
  g_spShellSink.Release();
 }
 PostQuitMessage(0);
}
```

We now have a program that displays all the Internet Explorer and Explorer windows, updates their locations as you navigate, and adds and removes them as new windows are created or existing ones are closed.

Reminder: This is a Little Program, which means that there is little to no error checking, and the design may be somewhat suboptimal. (For example, I use global variables everywhere because I'm lazy.) But it should give you enough of a head start so you can write a more robust version.

Exercise: There is still a subtle bug in `BuildWindowList`. Identify it and discuss how you would address it.

Raymond Chen

Follow