How do I monitor, or even control, the lifetime of an Explorer window?

devblogs.microsoft.com/oldnewthing/20110325-00

March 25, 2011

Raymond Chen

A customer wanted help with monitoring the lifetime of an Explorer window.

> We want to launch a copy of Explorer to open a specific folder, then wait until the user closes the folder before continuing. We tried launching a copy of Explorer with the folder on the command line, then doing a `WaitForSingleObject` on the process handle, but the wait sometimes completes immediately without waiting. How do we wait until the user closes the Explorer window?

This is another case of solving a problem halfway and then having trouble with the other half.

The reason that `WaitForSingleObject` returns immediately is that Explorer is a single-instance program (well, limited-instance). When you open an Explorer window, the request is handed off to a running copy of Explorer, and the copy of Explorer you launched exits. That's why your `WaitForSingleObject` returns immediately.

Fortunately, the customer was willing to explain their underlying problem.

> We have a wizard that creates some files in a directory based on information provided by the user, and we want to launch Explorer to view that directory so users can verify that things are set up the way they want them. When users close the Explorer window, we ask them if everything was good; if not, then we back up and let the user try again.

Aha, the program is using Explorer as a "view this folder for a little while" subroutine. Unfortunately, Explorer doesn't work that way. For example, the user might decide to use the Address Bar and go visit some other folders completely unrelated to your program, and your program would just be sitting there waiting for the user to close that window; meanwhile, the user doesn't realize that your program is waiting for it.

What you can do is host the Explorer Browser control inside a page of your wizard and control it with interfaces like `IExplorerBrowser`. You can disable navigation in the Explorer Browser (so the user can look only at the folder you want to preview), and the user can click *Back* if they want to try again or *Next* if they are happy and want to continue. This has the

additional advantage of keeping all the parts of your wizard inside the wizard framework itself, allowing users to continue using the wizard navigation model that they are already familiar with.

A sample program which uses the Explorer Browser control can be found in the Platform SDK.

For the impatient, here's the scratch program version. Note that this is the minimal version; in real life, you would probably want to set some options and stuff like that.

```
#include <shlobj.h>
IExplorerBrowser *g_peb;
void
OnSize(HWND hwnd, UINT state, int cx, int cy)
{
    if (g_peb) {
        RECT rc = { 0, 0, cx, cy };
        g_peb->SetRect(NULL, rc);
    }
}
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
    BOOL fSuccess = FALSE;
    RECT rc;
    PIDLIST_ABSOLUTE pidl = NULL;
    if (SUCCEEDED(CoCreateInstance(CLSID_ExplorerBrowser, NULL,
                        CLSCTX_INPROC, IID_PPV_ARGS(&g_peb))) &&
        GetClientRect(hwnd, &rc) &&
        SUCCEEDED(g_peb->Initialize(hwnd, &rc, NULL)) &&
        SUCCEEDED(SHParseDisplayName(
                        L"C:\\Program Files\\Internet Explorer",
                                    NULL, &pidl, 0, NULL)) &&
        SUCCEEDED(g_peb->SetOptions(EBO_NAVIGATEONCE)) &&
        SUCCEEDED(g_peb->BrowseToIDList(pidl, SBSP_ABSOLUTE))) {
        fSuccess = TRUE;
    }
    ILFree(pidl);
    return fSuccess;
}
void
OnDestroy(HWND hwnd)
{
    if (g_peb) {
        g_peb->Destroy();
        g_peb->Release();
    }
    PostQuitMessage(0);
}
```

This same technique of hosting the Explorer Browser control can be used for other types of "build your own burrito" scenarios: For example, you might host the Explorer Browser control in a window and tell users to copy files into that window. When they click OK or Next or whatever, you can enumerate the contents of the folder and do your business.

Armed with this knowledge, you can answer these customers' questions:

> We have found that the process state of Explorer.exe changes to signaled *before* the process terminates. Here's a sample program:
>
> ```
> int _tmain(int argc, TCHAR **argv)
> {
>  STARTUPINFO si = { sizeof(si) };
>  PROCESS_INFORMATION pi;
>  if (CreateProcess(TEXT("C:\\Windows\\Explorer.exe"), TEXT(" /e,C:\\usr"),
>                    NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
>   WaitForSingleObject(pi.hProcess);
>   CloseHandle(pi.hProcess);
>   CloseHandle(pi.hThread);
>  }
>  return 0;
> }
> ```
>
> If we change "Explorer.exe" to "Notepad.exe" then the process handle is signaled after Notepad terminates, as expected.

> We have a 32-bit shell extension for which a 64-bit version is not available. Since our clients are running 64-bit Windows, the 32-bit shell extension is not available in Explorer. How can we obtain access to this context menu?

> We have a shell extension that is not UAC-compliant. It requires that the user have administrative privileges in order to function properly. We would rather not disable UAC across the board just for this one shell extension. Is there a workaround that lets us run Explorer elevated temporarily?

Bonus sample program: The Explorer Browser Search Sample shows how to filter the view.

Bonus alternative: If you really just want to watch Explorer windows rather than host one, you can use the ShellWindows object, something I covered many years ago (and followed up with a much shorter scripting version).

Raymond Chen

Follow