Using an intermediate library to make the main library retargetable

devblogs.microsoft.com/oldnewthing/20150904-00

September 4, 2015

Raymond Chen

A customer was developing a static library targetting both Windows XP Win32 applications and universal Windows apps. (This was before Windows XP reached end-of-life.)

> Our library uses critical sections, but unfortunately there is no version `InitializeCriticalSection` that is available to both Windows XP Win32 applications and universal Windows apps. Universal Windows apps must use `InitializeCriticalSectionEx`, but that function is not available to Windows XP Win32 applications. Is there a way to dynamically target both Windows XP Win32 applications and universal Windows apps, pass WACK validation, and still have one library?
>
> We thought we could use `GetModuleHandle` and `GetProcAddress` to detect which platform we are one, but `GetModuleHandle` is not allowed in universal Windows apps, so we're back where we started.
>
> Are we stuck having two versions of our library, one for Windows XP Win32 applications and one for universal Windows apps?

Runtime dynamic linking (`LoadLibrary`, `GetProcAddress`) is not permitted in universal Windows apps, which means that for universal Windows apps, you must have an entry for `InitializeCriticalSectionEx` in your import table. But if that function is in your input table, then it won't load on Windows XP.

(You might think that you could have a second library to be used by Windows XP clients that implements the `InitializeCriticalSectionEx` function. Unfortunately, you will run afoul of dllimport.)

You are going to have to have separate libraries at some point, but you don't have to have two versions of your library. You could build your library to call, say, `ContosoInitializeCriticalSection`, and have two helper libraries, one for Windows XP Win32 applications and one for universal Windows apps, each of which implement the `ContosoInitializeCriticalSection` function in a manner appropriate to the target.

In other words, people targeting Windows XP would link to `ContosoCore.dll` and `ContosoXPSupport.dll`. People writing universal Windows apps would link to `ContosoCore.dll` and `ContosoStoreSupport.dll`.

This approach has a few advantages:

- It's simple, works (because it's so simple), and everybody understands it.
- All the files in your core library need to be compiled only once.

The second clause pays off if your library is large, or if you need to add new operating system targets.

Update: I guess I didn't make it clear. My suggestion is that `ContosoCore.dll` link to the nonexistent `ContosoSupport.dll`. If your program targets Windows XP, then rename `ContosoXPSupport.dll` to `ContosoSupport.dll`. If your program is a universal Windows app, then rename `ContosoStoreSupport.dll` to `ContosoSupport.dll`.

This technique also works with static libraries. You have a single `ContosoCore.lib` which calls a `ContosoInitializeCriticalSection` function. There are two implementations of `ContosoInitializeCriticalSection`, one in `ContosoXPSupport.lib` and another in `ContosoStoreSupport.lib`. Each application chooses which support library to link in.

Raymond Chen

Follow