Diagnosing a problem with calling conventions

devblogs.microsoft.com/oldnewthing/20040706-00

July 6, 2004

Raymond Chen

A commenter asks for help with an unresolved external. One of my goals is to give you more insight into how things work so you can solve problems yourself. This particular problem – resolving the error "Undefined symbol: '__stdcall(0) pl_pvcam_init (_pl_pvcam_init@0)' referenced from '_main' in Acquisition.c:15" is one example of something you can solve with the tips you've already learned.

First, let's look at the unresolved external itself. "_pl_pvcam_init@0". From the article this comment was posted to, you can see that the leading underscore and trailing @0 indicate that the function uses the __stdcall calling convention. (This is confirmed by the linker's undecoration of the name.)

So your function "_main" wants the function pl_pvcam_init with the __stdcall calling convention. But it's not found in the library even though you linked to it.

If you look inside the library itself, you'll find the desired symbols with some decoration. Decode that decoration. (My psychic powers tell me that when you do, you'll find that the decoration is "_pl_pvcam_init", which is the __cdecl calling convention.)

So now you see the problem. Your code is calling with the __stdcall calling convention, but the function actually uses the __cdecl calling convention. The calling conventions don't match up, so the linkage fails.

The solution, of course, is to fix the declaration of the pl_pvcam_init function in the header file to specify the correct calling convention. My psychic powers tell me that the header file doesn't specify any calling convention at all, which puts it at the mercy of the ambient calling convention for your project, which appears to be __stdcall. But the author of the header file expected __cdecl to be the default calling convention.

Put explicit calling conventions on the functions and you should be all set.

Raymond Chen

Follow