Studying linker error messages to find the cause of the unresolved external: Character sets

devblogs.microsoft.com/oldnewthing/20210701-00

July 1, 2021

Raymond Chen

A customer was encountering an unresolved external linker error when trying to link a plug-in with a static library.

```
// header.h
#define UNICODE
#define _UNICODE
#include <windows.h>

BOOL SetSessionName(LPCTSTR name);
```

This header file is used both by the implementation:

```
// implementation.cpp
#include "pch.h"
#include <header.h>

BOOL SetSessionName(LPCTSTR name)
{
    /* ... */
}
```

And it is used by the plug-in:

```
// plugin.cpp
#include "pch.h"
#include <header.h>

void Initialize()
{
    SetSessionName(TEXT("Fred"));
}
```

The result when building the plug-in is an unresolved external:

```
LNK2019 unresolved external "int __cdecl SetSessionName(char const*)" (?
SetSessionName@@YAHPBD@Z)
```

What's going on? The header file sets Unicode as the default before including the `windows.h` header file, so everything should be Unicode, shouldn't it?

Let's look at what the error message is telling us. It says that the plug-in wants a function that takes a `char const*`, which is what `LPCTSTR` maps to when ANSI is the default. So somehow the Unicode setting isn't sticking when the plug-in is using it.

I usd my psychic powers to guess that the plug-in had already performed its own `#include <windows.h>` before including `header.h`, and that initial inclusion of `windows.h` was done with ANSI as the default character set. The `header.h` is changing the character set too late.

Okay, so now that we understand the problem, how do we solve it?

One option is to give up on ANSI. Just be all-Unicode all the time. After all, any plug-in that is ANSI-based is going to have problems with file names, user names, all sorts of things.

But the customer said that they wanted to support both ANSI and Unicode plug-ins. Mind you, I'm not sure I believe that, seeing as their header file tried to `#define UNICODE` to force Unicode, but I'm going to take them at their word. Maybe the `#define UNICODE` was just an experiment.

If you want to support either character set, then you need to define two versions, one for each character set. (While you're at it, specify a calling convention already.) Classic Win32 uses C-style bindings, so you would have to decorate the function names manually:

```
EXTERN_C BOOL WINAPI SetSessionNameA(LPCSTR name);
EXTERN_C BOOL WINAPI SetSessionNameW(LPCWSTR name);
```

Your implementation file would have to implement both the A and W versions.

Another option is to use C++ decoration and overloads.

```
BOOL WINAPI SetSessionName(LPCSTR name);
BOOL WINAPI SetSessionName(LPCWSTR name);
```

The downside of this is that it requires the plug-in to use the same compiler that your framework is written in. This is generally not a great idea, since your customers will probably have a preferred toolchain, and forcing them to use a specific compiler (and perhaps even a specific version of a specific compiler) will make it harder for you to make friends.

Even worse: If your toolchain is the Microsoft Visual C compiler, then you have to deal with the `/Zc:wchar_t` option, which means bringing `__wchar_t` into the picture.

```
BOOL WINAPI SetSessionName(LPCSTR name);
BOOL WINAPI SetSessionName(unsigned short const* name);
BOOL WINAPI SetSessionName(__wchar_t const* name);
```

You now have to implement three versions of the function, although you presumably could have them all be wrappers around a common helper that takes `LPCWSTR`.

Related reading: Diagnosing a problem with calling conventions.

Raymond Chen

Follow