Undecorating names to see why a function can't be found

devblogs.microsoft.com/oldnewthing/20081229-00

December 29, 2008

Raymond Chen

Here's a problem inspired by actual events.

> When I build my project, it compiles fine, but it fails during the link step with an unresolved external:
>
> ```
> program.obj : error LNK2001: unresolved external symbol
> "public: virtual wchar_t const * __thiscall
> UILibrary::PushButton::GetName(class UILibrary::StringHolder * *)"
> (?GetName@PushButton@UILibrary@@UAEPB_WPAPAVStringHolder@2@@Z)
> ```
>
> The function I'm trying to call exists in the source code for `uilibrary.lib`; I'm looking at it right now. And the definition in the source code matches the declaration in the header file:
>
> ```
> namespace UILibrary {
>  …
>  class PushButton {
>  public:
>   virtual LPCWSTR GetName(StringHolder **Holder);
>  };
>  …
> }
> ```
>
> Why can't the linker find it? (Other functions in `uilibrary.lib` link just fine.)

In order to find something, you have to be looking in the right place, and the thing you're looking for actually needs to be there. (And you have to be able to see it when it's there.) The first part, looking in the right place, appears to be addressed by the parenthetical: The linker is definitely looking in `uilibrary.lib` since it managed to find other things in that library.

Let's look at the second step, then. Is the thing you're looking for really there? I fired up a little hex editor on `uilibrary.lib`, but you could use `strings` or, if you really want to get fancy, `link /dump /headers`. I went looking for "GetName@PushButton" to see if the member function was actually in the library.

And yup, the function is there. But it looks slightly different: `?GetName@PushButton@UILibrary@@UAEPBGPAPAVStringHolder@2@@Z`. (See if you can spot the difference.) Aha, the symbol couldn't be found because it indeed doesn't exist! What does exist is something that superficially resembles the symbol we want, but which has different decoration. We ask the `undname` program to convert this name into something a bit more readable:

```
C:\> undname ?GetName@PushButton@UILibrary@@UAEPBGPAPAVStringHolder@2@@Z
public: virtual unsigned short const * __thiscall
UILibrary::PushButton::GetName(class UILibrary::StringHolder * *)
```

Looking carefully at the two functions, we see that the difference is that the one that `program.obj` is looking for has a return type of `wchar_t const *`, whereas the one in the library returns a `unsigned short const *`.

At this point the answer is obvious. The library was compiled with the `/Zc:wchar_t-`* flag, which disables `wchar_t` as a native type. When that happens, the Windows header files gives the `wchar_t` symbol the definition `typedef unsigned short wchar_t;` On the other hand, the customer's project was being compiled without that switch, in which case `wchar_t` is a native type and not an alias for `unsigned short`.

Now you know enough to solve this customer's problem, which is very similar to the previous one:

> When I build my project, it compiles fine, but it fails during the link step with an unresolved external:
>
> ```
> program.obj : error LNK2019: unresolved external symbol
> "long __cdecl UILibrary::Initialize(bool)"
> (?Initialize@UILibrary@@YAJ_N@Z)
> ```

The function as it exists in the library undecorates as follows:

```
long __stdcall UILibrary::Initialize(bool)
```

Note

The `undname` program and the `/Zc:wchar_t-` switches are specific to the Microsoft Visual C++ compiler. Naturally, if you use a different compiler, you should use the utility or command line switch appropriate to your compiler. In particular, if you use the Visual Studio development environment, I'm told (but have not tried it myself) that the switch you're looking for is called "Treat wchar_t as a built-in type" on the "C/C++ Language" property page.

Raymond Chen

Follow