Understanding the classical model for linking: Sometimes you don't want a symbol to come along for a ride

devblogs.microsoft.com/oldnewthing/20130110-00

January 10, 2013

Raymond Chen

Continuing our study of the classical model for linking, let's take another look at the trick of taking symbols along for the ride.

The technique of taking symbols along for the ride is quite handy if that's what you want, but sometimes you don't actually want it. For example, a symbol taken along for the ride may create conflicts or create unwanted dependencies.

Here's an example: Suppose you have a library called `stuff.lib` where you put functions that are used by various modules in different projects. One of the files in your library might look like this:

```
// filedatestuff.cpp


BOOL GetFileCreationTimeW(
        LPCWSTR pszFile,
        FILETIME *pft)
{
    WIN32_FILE_ATTRIBUTE_DATA wfad;
    BOOL fSuccess = GetFileAttributesExW(pszFile,
                            GetFileExInfoStandard,
                            &wfad);
    if (fSuccess) {
        *pft = wfad.ftCreationTime;
    } else {
        pft->dwLowDateTime = 0;
        pft->dwHighDateTime = 0;
    }
    return fSuccess;
}


BOOL GetFileCreationTimeAsStringW(
         LPCWSTR pszFile,
         LPWSTR pszBuf,
         UINT cchBuf)
{
    FILETIME ft;
    BOOL fSuccess = GetFileCreationTimeW(pszFile, &ft);
    if (fSuccess) {
        fSuccess = SHFormatDateTimeW(&ft, NULL,
                                     pszBuf, cchBuf) > 0;
    }
    return fSuccess;
}
```

Things are working out great, people like the helper functions in your library, and then you get a bug report:

> When my program calls the `GetFileCreationTimeW` function, I get a linker error: `unresolved external: __imp__SHFormatDateTimeW`. If I remove my call to `GetFileCreationTimeW`, then my program builds fine.

You scratch your head. "The program is calling `GetFileCreationTimeW`, but that function doesn't call `SHFormatDateTimeW`, so why are we getting an unresolved external error? Any why hasn't anybody else run into this problem before?"

First question first. Why are we getting an unresolved external error for a nonexistent external dependency?

Because the `GetFileCreationTimeAsStringW` function got *taken along for the ride*. When the customer's program called `GetFileCreationTimeW`, that pulled in the `filedatestuff.obj` file, and that OBJ file contains both `GetFileCreationTimeW` *and* `GetFileCreationTimeAsStringW`. Since they are in the same OBJ file, pulling in one function pulls in all of them.

The fix is to split the `filedatastuff.cpp` file into two files, one for each function. That way, when you pull in one function, nobody else comes along for the ride.

Now to the second half of the question: Why did nobody run into this problem before?

The `GetFileCreationTimeW` function has a dependency on `GetFileAttributesExW`, which is a function in `KERNEL32.DLL`. On the other hand, the `GetFileCreationTimeAs-StringW` function has a dependency on `SHFormatDateTimeW`, which is a function in `SHLWAPI.DLL`. If somebody lists `KERNEL32.LIB` as a dependent library in their project, but they don't include `SHLWAPI.LIB` on that list, then they will encounter this problem because the linker will pull in the reference to `SHFormatDateTimeW` and have no way of resolving it.

Nobody ran into this before because `SHLWAPI.LIB` has lots of cute little functions in it, so most people include it in their project. Only if somebody is being frugal and leaving `SHLWAPI.LIB` out of their project will they run into this problem.

Bonus chatter: The suggestion to split the file into two will work, but if you are really clever, you can still do some consolidation. Instead of splitting up files by functional group (for example, "all `FILETIME` functions"), you need to split them up based on their dependencies ("functions that are dependent solely on `SHLWAPI.LIB`"). Of course, this type of organization may make the code harder to follow ("Why did you put `GetFileCreation-TimeAsStringW` and `HashString` in the same file?"), so you have to balance this against maintainability and readability. For example, somebody who is not aware of the classical model for linking may add a function to the file that has a dependency on `SHELL32.DLL`, and now your careful separation has fallen apart.

Raymond Chen

Follow