It's fine to rename a function in your DEF file, but when you do, you have to link to that function by its new name

devblogs.microsoft.com/oldnewthing/20100118-00

January 18, 2010

Raymond Chen

Jeffrey Riaboy asks why, if he renames a function in his DEF file, attempts to link to the function by its old name fail.

Well, um, yeah, because you renamed it.

Let's take the situation apart a bit; maybe it'll make more sense. I'm going to ignore a lot of details (`dllimport/dllexport` , calling conventions) since they are not relevant to the discussion and would end up just being distracting. I'm also going to assume we are running on an x86-class machine, just for concreteness. The same discussion works for other platforms; you just have to adjust the conventions accordingly.

First, here is some source code for a DLL, let's call it `FRED.DLL` :

```
int Dabba()
{
  return 0;
}
int Doo()
{
  return 1;
}
```

And here is the DEF file for `FRED.DLL` :

```
EXPORTS
 Yabba=Dabba
 Dabba=Doo
```

When you compile this DLL, the result will be something like this:

```
FRED.DLL:
 Yabba -> return 0;
 Dabba -> return 1;
```

The function exported as `Yabba` returns `0` because the DEF file said, “I want to export a function with the exported name `Yabba`; when somebody calls the function, I want control to go to the function I called `Dabba` internally.”

Similarly, the function exported as `Dabba` returns `1` because the DEF file said, “I want to export a function with the exported name `Dabba`; when somebody calls the function, I want control to go to the function I called `Doo` internally.”

Remember that symbolic information disappears during linking. The names of the functions and variables in the original source code are not stored anywhere in the DLL. The names exist only so that the linker can resolve symbolic references between object files. Once that’s done, the names are discarded: Their work is done. (See The classical model for linking for a discussion of how linking works under the classical model.)

Exported functions are also a mapping between labels and functions, but this mapping is not used when linking the DLL; rather, it is just a table the linker produces under the direction of your `DEF` file. To reduce confusion for the programmer writing the DLL, the name in the exported function table usually matches the name in the object files, but that is merely a convention. An entry in the export table that doesn’t perform renaming is just a shorthand for “I would like the exported name for this function to be the same as its internal name.” It’s a convenient typing-saver.

By analogy, Microsoft employees have one email address for use inside the company, and a different email address for use outside the company. Some employees choose to have their external email address be the same as their internal one, but that is hardly a requirement.

Meanwhile, the import library for our DLL looks something like this:

```
FRED.LIB:
 __imp__Yabba -> FRED.Yabba
 __imp__Dabba -> FRED.Dabba
 _Yabba@0 -> jmp [__imp__Yabba]
 _Dabba@0 -> jmp [__imp__Dabba]
```

As we saw before, each exported function results in two symbols in the import library, one with `__imp_` prepended to the exported name, which represents the import table entry, and one containing a stub function for the benefit of a naïve compiler.

Now let’s look at a program that wants to call some functions from `FRED.DLL`:

```
int Flintstone()
{
 Yabba();
 Dabba();
 Doo();
}
```

Let's say that these functions were not declared as `dllimport`, just for the sake of concreteness. (The discussion works the same if they were declared as `dllimport`, making the appropriate changes to the symbol names.) When the linker goes to resolve the call to `Yabba@0`, it will find the entry in `FRED.DLL` that says, "I've got a function called `Yabba@0`; the code for it is the single instruction `jmp [__imp__Yabba]`." When the program calls this function, the `jmp` instruction will jump through the import table entry for `FRED.Yabba`, which will wind up at the function in `FRED.DLL` exported under the name `Yabba`. If we look inside `FRED.DLL`, we see that this is a function that returns `0` (because it is the function which was called `Dabba` in the original source code, although that information was lost a long time ago).

Similarly, when the linker resolves the call to `Dabba@0`, it finds the entry in `FRED.DLL` which pulls in the one-line stub function which jumps through the import table entry for `Dabba@0`. This leads to a function that returns `1`, a function which was called `Doo` in the original source code.

However, that last call to `Doo` raises a linker error because it cannot find a function called `Doo` in the `FRED.LIB` import library. That's just the internal name for a function in the source code for `FRED.DLL`, a name which was lost during linking. If you want to call the function which had been called `Doo` in the original source code, you have to import it by its new name, `Dabba`.

In Jeffrey's case, he took a function which was internally referred to by a decorated name (`?Dispose@MyClass@@QAEAAV1@XZ`) and renaming it to an undecorated name (`MC_Dispose`). But when other modules tried to use the library, they got the error saying that "?Dispose@MyClass@@QAEAAV1@XZ" is not found. Which is correct: `?Dispose@MyClass@@QAEAAV1@XZ` was not found because it no longer exists under that name. You renamed it to `MC_Dispose`. Those modules need to link to the function `MC_Dispose` if they want to call the function "formerly known as ?Dispose@MyClass@@QAEAAV1@XZ".

Actually, Jeffrey's situation is more complicated than I described it because `?Dispose@MyClass@@QAEAAV1@XZ` undecorates to `public: class MyClass & __thiscall MyClass::Dispose(void)`; this is a method not a static function. I don't believe there's a way to override the name decoration algorithm for instance methods; the compiler is always going to generate a reference to `?Dispose@MyClass@@QAEAAV1@XZ`. So renaming the export doesn't buy you anything because you don't control the name on the import side.

Raymond Chen

Follow