Why do operating system files still adhere to the old 8.3 naming convention?

devblogs.microsoft.com/oldnewthing/20070402-00

April 2, 2007

Raymond Chen

Commenter Brian Reiter asks a duplicate of a question that was already submitted to the Suggestion Box: Darren asks why operating system† files still (for the most part) adhere to the old 8.3 naming convention.

There are a few reasons I can think of. I'm not saying that these are *the* reasons; I'm just brainstorming.

First, of course, the name of a DLL cannot change once it has been chosen, because that would break programs which linked to that DLL by its old name. Windows 95 did not require the system volume and user profile volume to support long file names, although that was certainly the case by default. Companies which used roaming profiles or redirected folders may have had a heavy investment in servers which did not support long file names. Therefore, all system files on Windows 95 had to conform to the 8.3 naming convention.

I believe that Windows NT permitted the system volume to be a short-file-names-only FAT partition as late as Windows 2000. Therefore, any DLL that existed in the Windows 2000 era had to conform to the 8.3 naming convention.

Starting in Windows XP, long file names became mandatory, and a few system files such as `shellstyle.dll` waded tentatively into the long file name world. (The .NET Framework folks jumped in with both feet with their managed DLLs, but notice that their unmanaged DLLs like `mscoree.dll` still conform to 8.3.) But the waters in this world can be treacherous for operating system components.

First of all, you have to worry about the automatically-generated short name. Suppose the operating system setup program is copying the `shellstyle.dll` file, but there is already a file called `shellstuff.dll`. The short name for `shellstuff.dll` will probably be `SHELLS~1.DLL`, and therefore the short name for `shellstyle.dll` will likely be `SHELLS~2.DLL`. Now, this may not be a big deal, except that some programs like to hard-code a file's short name. (There are a lot of programs that assume that the Program Files directory is C:\PROGRA~1, for example.)

Furthermore, you can create confusion if the same DLL is loaded by both its short and long names, since the loader treats them as distinct:

```
#include <stdio.h>
#include <windows.h>
int __cdecl main(int argc, char **argv)
{
 printf("%p\n", LoadLibrary("SHELLS~1.DLL"));
 printf("%p\n", LoadLibrary("SHELLSTYLE.DLL"));
 return 0;
}
```

If you run this program, you will get something like this:

```
6F2C0000
00340000
```

Even though the two paths refer to the same DLL, the loader treats them as different, and you end up with two copies of the same DLL loaded into memory. Now things get confusing, since you now have two sets of global variables, and if two components both use `SHELLSTYLE.DLL` but one used the short name and the other the long name, things get exciting when those two components try to talk about what they think is the same thing.

It's like that time when I was a child and our family took a trip to Disneyland. Our parents put my brother and me on the gondola ride, and upon arrival at the other end, we were to go to the Autopia ride which was right next door. The plan was that our parents would meet us at the exit to Autopia. When my brother and I exited Autopia, we expected our parents to be waiting there for us, but they were nowhere to be seen. Sticking to the plan, we waited patiently for our parents to arrive. We sat there for what seemed like two hours (but which was probably much less), until eventually we decided that my brother would stay put and I would go looking around, at which point it didn't take long for me to find my father, who was walking around looking for us.

What went wrong? Well, the problem was that the map of Disneyland showed Autopia, but what the map didn't say was that there were **two** Autopia rides (and therefore two Autopia exits) right next to each other. My brother and I were waiting by one exit, and our parents were waiting by the other. Each of us thought the other party was simply late.

Similarly, if a DLL goes by multiple names, you can end up with two copies of it loaded into the process, with different components talking about different copies, unaware that they are talking about different things.

And one final reason I can think of for sticking with 8.3 file names for operating system DLLs is simply, "Well, that's the way we've always done it. All the problems with 8.3 names are well-understood and under control. If we switched to long file names, we'd end up

discovering a whole new set of problems. Why mess with something that works if it isn't broken?"

Better the devil you know.

Exercise: Why is it okay for the .NET Framework to use long file names for their managed DLLs?

Nitpicker's Corner

†s/operating system/Windows operating system/. Apparently nothing is obvious from context any more.

Raymond Chen

Follow