The evolution of version resources – corrupted 32-bit version resources

devblogs.microsoft.com/oldnewthing/20061222-00

December 22, 2006

Raymond Chen

Last time we looked at the format of 32-bit version resources, but I ended with the remark that what you saw purported to be the resources of `shell32.dll` but actually weren't. What's going on here?

The resources I presented last time were what the resources of `shell32.dll` **should have been**, but in fact they aren't.

A common mistake in generating 32-bit resources is to mistreat the `cbData` field of the structure I called a `VERSIONNODE` as a count of **characters** rather than a count of bytes if the type is Unicode text. Even Microsoft's own Resource Compiler has fallen into this trap! For example, consider this `VERSIONNODE` I presented last time:

```
0098  4C 00         // cbNode (node ends at 0x0088 + 0x004C = 0x00D40)
009A  2C 00         // cbData
009C  01 00         // wType = 1 (string data)
009E  43 00 6F 00 6D 00 70 00 61 00 6E 00 79 00 4E 00
      61 00 6D 00 65 00 00 00
                    // L"CompanyName" + null terminator
00B6  00 00         // padding to restore alignment
00B8  4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00
      74 00 20 00 43 00 6F 00 72 00 70 00 6F 00 72 00
      61 00 74 00 69 00 6F 00 6E 00 00 00
                    // L"Microsoft Corporation" + null terminator
00E4                // no padding needed
```

In real life, the data take the following form:

```
0098  4C 00         // cbNode (node ends at 0x0088 + 0x004C = 0x00D40)
009A  16 00         // cchData (!)
009C  01 00         // wType = 1 (string data)
...
```

These malformed version resources manage to get away without crashing too horribly because the standard format of version resources uses string data only in leaf nodes. Therefore, the incorrect `cbData` affects only the node itself and doesn't cause the child

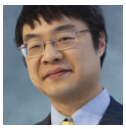
nodes to be parsed incorrectly (since there are no child nodes).

Until somebody tries to read, say, `\StringFileInfo\040904B0\CompanyName\oops`. After the `VerQueryValue` function locates the `VERSIONNODE` corresponding to `CompanyName`, it tries to locate the first child node and, due to the incorrect `cbData`, ends up misinterpreting the middle of the string as if it were the start of a child `VERSIONNODE`. Things only go downhill from there.

They're just lucky that nobody actually asks for that.

But wait, there's more. Somebody who calls the `VerQueryValueA` function expects to have the version string returned as ANSI, so `VerQueryValueA` needs to know how many characters to convert from Unicode to ANSI. If `VerQueryValue` trusted the erroneous `cbData` value, then ANSI callers would get only half the data they were expecting.

As a result of this mess, the `VerQueryValue` function keeps its eyes open and anticipates that the version resource it was given to parse may have been generated by one of these buggy version resource compilers and goes to some extra effort to accommodate those bugs.



Raymond Chen

Follow