The evolution of version resources – 16-bit version resources

devblogs.microsoft.com/oldnewthing/20061220-15

December 20, 2006

Raymond Chen

I return to the extremely sporadic series on resources with a description of the version resource. You don't need to know how version resources are formatted internally; you should just use the version resource manipulation functions `GetFileVersionInfo`, `VerQueryValue`, and their friends. I'm providing this information merely for its historical significance.

Version resources can be viewed as a serialized tree structure. Each node of the tree has a name and associated data (either binary or text), and each node can have zero or more child nodes. The root node is always named `VS_VERSION_INFO` and is a binary node consisting of a `VS_FIXEDFILEINFO` structure. Beyond that, you can call your nodes anything you want and give them any kind of data you want. But if you want other people to understand your version information, you'd be best off following the conventions I describe below. Actually, since people seem to prefer diagrams to words, I'll give you a diagram:

`VS_VERSION_INFO`	`VS_FIXEDFILEINFO` structure (binary)
`StringFileInfo`	(no data)
`xxxxyyyy`	(no data)
`CompanyName`	string for xxxxyyyy
`FileDescription`	string for xxxxyyyy
`FileVersion`	string for xxxxyyyy
`...`	
`zzzzwwww`	(no data)
`CompanyName`	string for zzzzwwww
`FileDescription`	string for zzzzwwww

`FileVersion`	string for zzzzwwww
`...`	
`VarFileInfo`	(no data)
`Translation`	array of locale/codepage pairs (binary, variable-size)

The child nodes can appear in any order, and the strings like `CompanyName` are all optional. `VarFileInfo\Translation`, however, is mandatory (by convention).

If you've used `VerQueryValue`, you know that the binary data stored under `VarFileInfo\Translation` consists of a variable-length array of locale/codepage pairs, each of which in turn corresponds to a child of the `StringfileInfo` node. I'm not going to go into what each of the strings means and how the local/codepage pairs turn into child nodes of `StringFileInfo`; I'll leave you to research that on your own (assuming you don't already know).

How does this tree get stored into a resource? It's actually quite simple. Each node is stored in a structure which takes the following form (in pseudo-C):

```
struct VERSIONNODE {
  WORD  cbNode;
  WORD  cbData;
  CHAR  szName[];
  BYTE  rgbPadding1[]; // DWORD alignment
  BYTE  rgbData[cbData];
  BYTE  rgbPadding2[]; // DWORD alignment
  VERSIONNODE rgvnChildren[];
};
```

In words, each version node begins with a 16-bit value describing the size of the nodes in bytes (including its children), followed by a 16-bit value that specifies how many bytes of data (either binary or text) are associated with the node. (If the node contains text data, the count includes the null terminator.) Next comes the null-terminated name of the node and padding bytes to bring us back into `DWORD` alignment. After the key name (and optional padding) comes the data, again followed by padding bytes to bring us back into `DWORD` alignment. Finally, after all the node information come its children.

Since each of the children might themselves have children, you can see how the tree structure "flattens" into this serialized format. To move from one node to its next sibling, you skip ahead by `cbNode` bytes. To move from a node to its first child, you skip over the key name and associated data.

Let's take a look at the resources for the 16-bit `shell.dll` to see how this all fits together.

```
0000  E4 01 34 00 56 53 5F 56-45 52 53 49 4F 4E 5F 49  ..4.VS_VERSION_I
0010  4E 46 4F 00 BD 04 EF FE-00 00 01 00 0A 00 03 00  NFO.............
0020  67 00 00 00 0A 00 03 00-67 00 00 00 3F 00 00 00  g.......g...?...
0030  0A 00 00 00 01 00 01 00-02 00 00 00 00 00 00 00  ................
0040  00 00 00 00 00 00 00 00-78 01 00 00 53 74 72 69  ........x...Stri
0050  6E 67 46 69 6C 65 49 6E-66 6F 00 00 64 01 00 00  ngFileInfo..d...
0060  30 34 30 39 30 34 45 34-00 00 00 00 27 00 17 00  040904E4....'...
0070  43 6F 6D 70 61 6E 79 4E-61 6D 65 00 4D 69 63 72  CompanyName.Micr
0080  6F 73 6F 66 74 20 43 6F-72 70 6F 72 61 74 69 6F  osoft Corporatio
0090  6E 00 00 00 2A 00 16 00-46 69 6C 65 44 65 73 63  n...*...FileDesc
00A0  72 69 70 74 69 6F 6E 00-57 69 6E 64 6F 77 73 20  ription.Windows
00B0  53 68 65 6C 6C 20 6C 69-62 72 61 72 79 00 00 00  Shell library...
00C0  16 00 06 00 46 69 6C 65-56 65 72 73 69 6F 6E 00  ....FileVersion.
00D0  33 2E 31 30 00 00 00 00-1A 00 06 00 49 6E 74 65  3.10........Inte
00E0  72 6E 61 6C 4E 61 6D 65-00 00 00 00 53 48 45 4C  rnalName....SHEL
00F0  4C 00 00 00 3B 00 27 00-4C 65 67 61 6C 43 6F 70  L...;.'.LegalCop
0100  79 72 69 67 68 74 00 00-43 6F 70 79 72 69 67 68  yright..Copyrigh
0110  74 20 A9 20 4D 69 63 72-6F 73 6F 66 74 20 43 6F  t . Microsoft Co
0120  72 70 2E 20 31 39 38 31-2D 31 39 39 36 00 00 00  rp. 1981-1996...
0130  22 00 0A 00 4F 72 69 67-69 6E 61 6C 46 69 6C 65  "...OriginalFile
0140  6E 61 6D 65 00 00 00 00-53 48 45 4C 4C 2E 44 4C  name....SHELL.DL
0150  4C 00 00 00 39 00 29 00-50 72 6F 64 75 63 74 4E  L...9.).ProductN
0160  61 6D 65 00 4D 69 63 72-6F 73 6F 66 74 AE 20 57  ame.Microsoft. W
0170  69 6E 64 6F 77 73 28 54-4D 29 20 4F 70 65 72 61  indows(TM) Opera
0180  74 69 6E 67 20 53 79 73-74 65 6D 00 00 00 00 00  ting System.....
0190  1A 00 06 00 50 72 6F 64-75 63 74 56 65 72 73 69  ....ProductVersi
01A0  6F 6E 00 00 33 2E 31 30-00 00 00 00 14 00 04 00  on..3.10........
01B0  57 4F 57 20 56 65 72 73-69 6F 6E 00 34 2E 30 00  WOW Version.4.0.
01C0  24 00 00 00 56 61 72 46-69 6C 65 49 6E 66 6F 00  $...VarFileInfo.
01D0  14 00 04 00 54 72 61 6E-73 6C 61 74 69 6F 6E 00  ....Translation.
01E0  09 04 E4 04                                      ....
```

We start with the root node.

```
0000  E4 01         // cbNode (node ends at 0x0000 + 0x01E4 = 0x01E4)
0002  34 00         // cbData = sizeof(VS_FIXEDFILEINFO)
0004  56 53 5F 56 45 52 53 49 4F 4E 5F 49 4E 46 4F 00
                    // "VS_VERSION_INFO" + null terminator
```

Notice that the size of the root node equals the size of the entire version resource. This is to be expected, of course, because the version resource is merely a serialization of the resource tree diagram.

Since the string name (plus null terminator) happens to come out to an exact multiple of four bytes, there is no need for padding between the name and the binary data, which takes the form of a `VS_FIXEDFILEINFO`:

```
0014  BD 04 EF FE   // dwSignature
0018  00 00 01 00   // dwStrucVersion
001C  0A 00 03 00   // dwFileVersionMS = 3.10
0020  67 00 00 00   // dwFileVersionLS = 0.103
0024  0A 00 03 00   // dwProductVersionMS = 3.10
0028  67 00 00 00   // dwProductVersionLS = 0.103
002C  3F 00 00 00   // dwFileFlagsMask
0030  0A 00 00 00   // dwFileFlags
0034  01 00 01 00   // dwFileOS = VOS_DOS_WINDOWS16
0038  02 00 00 00   // dwFileType = VFT_DLL
003C  00 00 00 00   // dwFileSubtype
0040  00 00 00 00   // dwFileDateMS
0044  00 00 00 00   // dwFileDateLS
```

The structure is also a multiple of 4 bytes in length, so no padding is necessary between the data and the child nodes.

```
0048  78 01         // cbNode (node ends at 0x0048 + 0x0178 = 0x01C0)
004A  00 00         // cbData (no data)
004C  53 74 72 69 6E 67 46 69 6C 65 49 6E 66 6F 00
                    // "StringFileInfo" + null
005B  00            // padding to restore alignment
005C                // no data
```

The first child is the `StringFileInfo`. It has no data, so its own children come directly after the name (and padding). And the children of `StringFileInfo` are the language nodes.

```
005C  64 01         // cbNode (node ends at 0x005C + 0x0164 = 0x01C0)
005E  00 00         // cbData (no data)
0060  30 34 30 39 30 34 45 34 00
                    // "040904E4" + null terminator
0069  00 00 00      // padding to restore alignment
006C                // no data
```

The children of the language node are the strings. This is where all the goodies can be found.

```
006C  27 00         // cbNode (node ends at 0x006C + 0x0027 = 0x0093)
006E  17 00         // cbData
0070  43 6F 6D 70 61 6E 79 4E 61 6D 65 00
                    // "CompanyName" + null terminator
007C                // no padding needed
007C  4D 69 63 72 6F 73 6F 66 74 20 43 6F
      72 70 6F 72 61 74 69 6F 6E 00
                    // "Microsoft Corporation" + null terminator
0091  00 00 00      // padding to restore alignment
```

Notice that the padding bytes are not counted in the `cbData`. In fact, the padding bytes at the end of the data don't even count towards the `cbNode`. This is a leaf node since we already reach the end of the node once we store the data. Therefore, the next node in the

version resource is a sibling, not a child.

```
0094  2A 00         // cbNode (node ends at 0x0094 + 0x002A = 0x00BE)
0096  16 00         // cbData
0098  46 69 6C 65 44 65 73 63 72 69 70 74 69 6F 6E 00
                   // "FileDescription" + null terminator
00A8               // no padding needed
00A8  57 69 6E 64 6F 77 73 20 53 68 65 6C 6C 20 6C 69
      62 72 61 72 79 00
                   // "Windows Shell library" + null terminator
00BE  00 00        // padding to restore alignment
```

All of these nodes have no children since we run out of bytes in `cbNode` after representing the node's name and data.

```
00C0  16 00         // cbNode (node ends at 0x00C0 + 0x0016 = 0x00D6)
00C2  06 00         // cbData
00C4  46 69 6C 65 56 65 72 73 69 6F 6E 00
                    // "FileVersion" + null terminator
00D0  33 2E 31 30 00
                    // "3.10"
00D5  00 00 00      // padding to restore alignment
00D8  1A 00         // cbNode (node ends at 0x00D8 + 0x001A = 0x00F2)
00DA  06 00         // cbData
00DC  49 6E 74 65 72 6E 61 6C 4E 61 6D 65 00
                    // "InternalName" + null terminator
00E9  00 00 00      // padding to restore alignment
00EC  53 48 45 4C 4C 00
                    // "SHELL" + null terminator
00F2  00 00         // padding to restore alignment
00F4  3B 00         // cbNode (node ends at 0x00F4 + 0x003B = 0x12E)
00F6  27 00
00F8  4C 65 67 61 6C 43 6F 70 79 72 69 67 68 74 00
                    // "LegalCopyright" + null terminator
0107  00            // padding to restore alignment
0108  43 6F 70 79 72 69 67 68 74 20 A9 20 4D 69 63 72
      6F 73 6F 66 74 20 43 6F 72 70 2E 20 31 39 38 31
      2D 31 39 39 36 00
                    // "Copyright © Microsoft Corp. 1981-1996"
                    // + null terminator + another null terminator?
012F  00            // padding to restore alignment
```

Wait a second, what's that "another null terminator"? if you count the bytes, you'll see that the `cbData` for the `LegalCopyright` node counts not only the terminating null, but another bonus null after it. I suspect that somebody put an extra null terminator in the resource file by mistake:

```
    VALUE "LegalCopyright", "Copyright\251 Microsoft corp. 1981-1996\0"
```

For whatever reason, there's an extra null in there.

```
0130  22 00         // cbNode (node ends at 0x0130 + 0x0022 = 0x0152)
0132  0A 00         // cbData
0134  4F 72 69 67 69 6E 61 6C 46 69 6C 65 6E 61 6D 65 00
                    // "OriginalFilename" + null terminator
0145  00 00 00      // padding to restore alignment
0148  53 48 45 4C 4C 2E 44 4C 4C 00
                    // "SHELL.DLL" + null terminator
0152  00 00         // padding to restore alignment
0154  39 00         // cbNode (node ends at 0x0154 + 0x0039 = 0x018D)
0156  29 00         // cbData
0158  50 72 6F 64 75 63 74 4E 61 6D 65 00
                    // "ProductName" + null terminator
0164  4D 69 63 72 6F 73 6F 66 74 AE 20 57 69 6E 64 6F
      77 73 28 54 4D 29 20 4F 70 65 72 61 74 69 6E 67
      20 53 79 73 74 65 6D 00 00
                    // "Microsoft® Windows(TM) "
                    // "Operating System" + null terminator
                    // + another null terminator?
018D  00 00 00      // padding to restore alignment
```

There's another of those extra null terminators. Go figure.

```
0190  1A 00         // cbNode (node ends at 0x0190 + 0x001A = 0x01AA)
0192  06 00         // cbData
0194  50 72 6F 64 75 63 74 56 65 72 73 69 6F 6E 00
                    // "ProductVersion" + null terminator
01A3  00            // padding to restore alignment
01A4  33 2E 31 30 00 00
                    // "3.10" + null terminator
                    // + another null terminator?
01AA  00 00         // padding to restore alignment
01AC  14 00         // cbNode (node ends at 0x01AC + 0x0014 = 0x01C0)
01AE  04 00         // cbData
01B0  57 4F 57 20 56 65 72 73 69 6F 6E 00
                    // "WOW Version"
01BC                // no padding needed
01BC  34 2E 30 00   // "4.0" + null terminator
01C0                // no padding needed
```

Once we reach offset `0x01C0`, we're reached the end of not only the `WOW Version` node, but also the end of the `040904E4` node and the `StringFileInfo` node. Therefore, the next node is a child of the root.

```
01C0  24 00         // cbNode (node ends at 0x01C0 + 0x0024 = 0x01E4)
01C2  00 00         // cbData (no data)
01C4  56 61 72 46 69 6C 65 49 6E 66 6F 00
                    // "VarFileInfo" + null terminator
01D0                // no padding needed
01D0                // no data
```

Since we have not reached the end of the `VarFileInfo` node, the data that comes next must be a child node.

```
01D0  14 00         // cbNode (noed ends at 0x01D0 + 0x0014 = 0x01E4)
01D4  04 00         // cbData
01D6  54 72 61 6E 73 6C 61 74 69 6F 6E 00
                    // "Translation" + null terminator
01E0  09 04 E4 04   // 0x0409 = US English
                    // 0x04E4 = 1252 = Western European
01E4                // no padding needed
```

And once we've reached offset `0x01E4`, we've reached the end of the `Translation` node, the `VarFileInfo` node, and the root node.

Thus, we have reconstructed the original version resource:

```
FILEVERSION    3,10,0,103
PRODUCTVERSION 3,10,0,103
FILEFLAGSMASK  VS_FFI_FILEFLAGSMASK
FILEFLAGS      VS_FF_PRERELEASE | VS_FF_PRIVATEBUILD
FILEOS         VOS_DOS_WINDOWS16
FILETYPE       VFT_DLL
FILESUBTYPE    VFT_UNKNOWN
BEGIN
 BLOCK "StringFileInfo"
 BEGIN
  BLOCK "040904E4"
  BEGIN
   VALUE "CompanyName", "Microsoft Corporation"
   VALUE "FileDescription", "Windows Shell library"
   VALUE "FileVersion", "3.10"
   VALUE "InternalName", "SHELL"
   VALUE "LegalCopyright", "Copyright\251 Microsoft corp. 1981-1996\0"
   VALUE "OriginalFilename", "SHELL.DLL"
   VALUE "ProductName", "Microsoft\256 Windows(TM) Operating System\0"
   VALUE "ProductVersion", "3.10\0"
   VALUE "WOW Version", "4.0"
  END
 END
 BLOCK "VarFileInfo"
 BEGIN
  VALUE "Translation", 0x0409, 0x04E4
 END
END
```

Next time, we'll look at how version resources are represented in 32-bit resources.

Raymond Chen

Follow