Random musings on the introduction of long file names on FAT

devblogs.microsoft.com/oldnewthing/20110826-00

August 26, 2011

Raymond Chen

Tom Keddie thinks that the format of long file names on FAT deserves an article. Fortunately, I don't have to write it; somebody else already did. So go read that article first. I'm just going to add some remarks and stories. Hi, welcome back. Coming up with the technique of setting Read-only, System, Hidden, and Volume attributes to hide LFN entries took a bit of trial and error. The volume label was the most important part, since that was enough to get 90% of programs which did low-level disk access to lay off those directory entries. The other bits were added to push the success rate ever so close to 100%. The linked article mentions rather briefly that the checksum is present to ensure that the LFN entries correspond to the SFN entry that immediately follows. This is necessary so that if the directory is modified by code that is not LFN-aware (for example, maybe you dual-booted into Windows 3.1), and the file is deleted and the directory entry is reused for a different file, the LFN fragments won't be erroneously associated with the new file. Instead, the fragments are "orphans", directory entries for which the corresponding SFN entry no longer exists. Orphaned directory entries are treated as if they were free. The cluster value in a LFN entry is always zero for compatibility with disk utilities who assume that a nonzero cluster means that the directory entry refers to a live file. The linked article wonders what happens if the ordinals are out of order. Simple: If the ordinals are out of order, then they are invalid. The file system simply treats them as orphans. Here's an example of how out-of-order ordinals can be created. Start with the following directory entries:

(2) "e.txt"
(1) "Long File Nam"
"LONGFI~1.TXT"
(2) "e2.txt"
(1) "Long File Nam"
"LONGFI~2.TXT"

Suppose this volume is accessed by a file system that does not support long file names, and the user deletes `LONGFI~1.TXT`. The directory now looks like this:

(2) “e.txt”
(1) “Long File Nam”
(free)
(2) “e2.txt”
(1) “Long File Nam”
“LONGFI~2.TXT”

Now the volume is accessed by a file system that supports long file names, and the user renames `Long File Name2.txt` to `Wow that's a really long file name there.txt`.

(2) “e.txt”
(4) “e.txt”
(3) “ile name ther”
(2) “really long f”
(1) “Wow that’s a “
“WOWTHA~1.TXT”

Since the new name is longer than the old name, more LFN fragments need to be used to store the entire name, and oh look isn’t that nice, there are some free entries right above the ones we’re already using, so let’s just take those. Now if you read down the table, you see that the ordinal goes from 2 up to 4 (out of order) before continuing in the correct order. When the file system sees this, it knows that the entry with ordinal 2 is an orphan. One last historical note: The designers of this system didn’t really expect Windows NT to adopt long file names on FAT, since Windows NT already had its own much-better file system, namely, NTFS. If you wanted long file names on Windows NT, you’d just use NTFS and call it done. Nevertheless, the decision was made to store the file names in Unicode on disk, breaking with the long-standing practice of storing FAT file names in the OEM character set. The decision meant that long file names would take up twice as much space (and this was back in the days when disk space was expensive), but the designers chose to do it anyway “because it’s the right thing to do.”

And then Windows NT added support for long file names on FAT and the decision taken years earlier to use Unicode on disk proved eerily clairvoyant.

Raymond Chen

Follow