The Old New Thing

A file can go by multiple names, but two files can't have the same name

8 Dec 2008 10:00 AM | 48

Thanks to short file names and hard links, a single file can go by multiple names. (And for the purpose of today's discussion, I'm treating the full path as the name instead of just the part after the last backslash. Don't make me bring back the nitpicker's corner.) For example, C:\PROGRA~1 and C:\Program Files are two possible names for the same directory thanks to short names. [Typo fixed 7:15am.] If you've created hard links, then you can give a single file two entirely unrelated names, and those names need not even be in the same directory.

On the other hand, you can't have two files with the same name. What would that even mean? Which one would you get if you issue an Open call? How would you open the other one? Heck, even before we get to this point: How do you even create two files with the same name? If you call `CreateFile` to create the "second" file, it'll just open the existing one!

That's why I am baffled by this question that asks:

> I've seen a few cases where people write their own version of GetLongPathName (usually because they need to support NT4) using FindFirstFile. Are there situations where that approach would return an incorrect path? Is it safe in practice because FindFirstFile("foo.bar") always will return the exact match first before returning "foo.barbaz"?

This question assumes that it's possible to have two files in a directory with the same name: One is the file "foo.bar"; the other is the file that goes by the long name "foo.barbaz" and the short name "foo.bar".

I'm not sure what the sequence of events would be that could result in two files with the same name. Here's one scenario:

1. Create file "foo.barbaz". It gets assigned the short name "foo.bar".
2. Create file "foo.bar". This creates a new file called "foo.bar" which conflicts with the short name of the existing file "foo.barbaz".

Except that's not what'll happen. When you perform step 2, the attempt to create the file "foo.bar" merely overwrites the existing file which has "foo.bar" as one of its names. Result: A single file with long name "foo.barbaz" and short name "foo.bar".

Another scenario might go like this:

1. Create file "foo.bar".
2. Create file "foo.barbaz". The file system auto-assigns the short name "foo.bar".

Except that's not what happens either. At the second step, the file system generates a short name like "foo~1.bar" specifically to avoid the name collision.

You can run these experiments yourself from the command line to confirm. The "dir /x" command will come in handy.

Blog - Comment List MSDN TechNet

Comments

Karellen
8 Dec 2008 10:07 AM
#

When I encounter such a failure of logic, I find it useful to quote Charles Babbage[0] and reply:

"I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question."

[0] http://en.wikiquote.org/wiki/Charles_Babbage

Adam
8 Dec 2008 10:08 AM

#

"For example, C:\PROGRA~1 and C:\Programs and Settings are two possible names for the same directory thanks to short names."

I'm assuming that's a typo? Or did Vista decide to merge "Documents and Settings" and "Program Files" (and "Program Files (x86)" for 64-bit)?

[*I meant Program Files, but the statement is still correct as written: C:\PROGRA~1 and C:\Programs and Settings are two possible names for the same directory. Perhaps I shouldn't have tried to use a real-life example. "C:\AWESOM~1 and C:\Awesome Videos" for example. -Raymond*]

Gabe
8 Dec 2008 10:22 AM
#

You can use POSIX (or just FILE_FLAG_POSIX_SEMANTICS) to effectively create two files with the same name (like FOO.BAR and foo.bar). Then only one of them will be accessible to regular Win32 programs.

SRS
8 Dec 2008 10:30 AM
#

Disclaimer: I am not James.

I think what James might mean is:

1. Create file "foo.barbaz". It gets assigned the short name "foo.bar" by the system.

2. Write a special implementation of GetLongPathName() (say MyGetLongPathName() that uses FindFirstFile() (A or W - I don't know).

3. Call MyGetLongPathName("foo.bar").

4. Should the result be "foo.bar" or "foo.barbaz"?

I think James wants the result to be "foo.bar", not "foo.barbaz", as "foo.bar" is an exact match for the input.

[*That can't be the original question, because if FindFirstFile behaved that way, you couldn't use it to write MyGetLongPathName: whatever you passed in would come right back out; you'd never get the long version. -Raymond*]

Mark (The other Mark)
8 Dec 2008 10:39 AM
#

He's asking if a directory contains Foo.bar and Foo.barbaz, if FindFirstFile("Foo.Bar") will always return information about Foo.bar.

He's probably basing this on "The FindFirstFile function opens a search handle and returns information about the first file that the file system finds with a name that matches the specified pattern. This may or may not be the first file or directory that appears in a directory-listing application (such as the dir command) when given the same file name string pattern."

If you consider ("Foo.Bar") as an implied ("Foo.Bar*"), his question makes sense. Kindof.

JCR
8 Dec 2008 10:44 AM
#

Raymond:

In the first scenario, doesn't foo.barbaz get assigned a short name of FOO~1.BAR. At least that is the result on Win XP.

[*Try a non-Microsoft filesystem. -Raymond*]

Michael
8 Dec 2008 11:17 AM
#

In Unix there are no short names, so foo.bar and foo.barbaz can both exist without the ~1 short name convention. Now, what happens if a Windows client is connected to a Samba share from a Unix machine, and the remote share has foo.bar and foo.barbaz? What will FindFirstFile("foo.bar") do in this case? Does Windows assign short names to these files for its own use when the share is accessed?

Steve Thresher
8 Dec 2008 11:33 AM
#

Slightly off topic but can anyone tell me why you cannot create a file with a name that starts with one of the reserved filenames? eg. c:\temp\com2.txt

Leo Davidson
8 Dec 2008 11:39 AM
#

Michael: I think that's up to Samba rather than Windows. I'm not sure what Samba actually does in this case but Samba presents itself to Windows as an SMB network share with Win32 semantics and Windows won't do anything special for it.

Strange things can and do happen as a result of Samba having to pretend that the Unix filesystem it is exposing is really a Win32 filesystem. It's unavoidable.

I encountered a performance problem with one Samba share where creating files took several seconds per file. It turned out that (that version and configuration of) Samba was re-reading the directory for each file creation request, checking any file names conflicted, ignoring case. Creating a file is usually logarithmic time (with respect to the number of files in the directory) but this made it linear time. You can imagine the performance and scaling problems that resulted.

Not really the fault of Samba or Windows (and I believe Samba has options to make the problem less painful); just one of those things. Stuff that we normally take for granted can behave very differently when something tries to emulate another quite different thing.

(NTFS can have case-sensitive volumes and/or volumes with short names switched off... It'll confuse a lot of Win32 software, though.)

Leo Davidson
8 Dec 2008 11:41 AM
#

Steve Thresher: Raymond talked about that in a previous post. Here you go:

http://blogs.msdn.com/oldnewthing/archive/2003/10/22/55388.aspx

Alexandre Grigoriev
8 Dec 2008 12:14 PM
#

Gabe & Leo Davidson,

Win32 is ignoring FILE_FLAG_POSIX_SEMANTICS. It doesn't matter if you pass it to CreateFile.

Steve Thresher:

try:

copy a.txt \\?\c:\temp\com2.txt

These special legacy mappings are ignored if you use CreateFileW (as cmd.exe does) and the name is prepended with \\?\

Yuhong Bao
8 Dec 2008 12:23 PM
#

"Win32 is ignoring FILE_FLAG_POSIX_SEMANTICS. It doesn't matter if you pass it to CreateFile."

Have you heard of the ObCaseInsensitive registry key?

Gabe
8 Dec 2008 12:32 PM
#

Alexandre Grigoriev: I'm pretty sure that Win32 doesn't do anything to strip the POSIX flag, otherwise why would it be in the documentation? The kernel itself will ignore any case-sensitivity if HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\kernel\obcaseinsensitive set to 1, however.

You probably have that set, as it is now the default probably for security reasons. Allowing case-sensitivity allows malware to create files that the user can't access or delete, so it is now opt-in instead of opt-out.

Neil
8 Dec 2008 12:46 PM
#

On Netware 4 the default short name was simply 8.3 truncation, so "foo.barbaz" would have a short name of "foo.bar", although of course that isn't relevant to the FindFirstFile question. I did see a problem related to this though, which is less likely on systems with better short path name generation:

1. Create a file foo~1.bar (or whatever the default short filename is for foo.barbaz on your target filesystem)

2. Create a file foo.barbaz (which now gets a short filename of foo~2.bar)

3. Try to copy both files to a new folder.

Depending on the application, it could copy foo.barbaz first, giving it a short filename of foo~1.bar, then wonder what to do with foo~1.bar ...

Joshua Muskovitz
8 Dec 2008 12:54 PM
#

This was also covered (somewhat) in http://blogs.msdn.com/oldnewthing/archive/2005/07/20/440918.aspx.

We recently encountered a bug in our app (ok, a bug in all of this backward compatibility hell, but we were caught in the line of fire) where a file with a long name of "20080331-BOST.Csv" was found with the wildcard "2007*.Csv". Needless to say, this caused a lot of head scratching until it occurred to us to dir /x it:

```
dir /x

10/02/2008  12:09 PM                56 200709~1.CSV 20080331-BOST.Csv
```

The really freaky part of it was that this evil file was shipped to us in a single zip, along with many other files which behaved properly. But the problem only occurred if you unzipped the *entire* archive. If you just grabbed the "bad" files, they worked fine.

Some quick reverse entineering determined that the algorithm Windows uses to generate the non-colliding short filename starts by using the first six characters of the long name + tilde + a digit, but when that namespace runs out (after only ~4? huh?), it moves on to an even more esoteric algorithm, namely the first two characters + a four-digit hex hash of the long filename + tilde + a digit. (We never determined what happens when *that* namespace fills up.)

In our case, it was because the earlier unzipped files filled the namespace, forcing the hex hash method, and the hash happened to be 0x0709 for this particular filename.

We opted not to attempt to "fix" this problem.

Kristofer
8 Dec 2008 1:38 PM
#

I think the original asker may have assumed it would be making a substring search rather than an exact search. It's not necessarily the case that they thought two files could have the same name, and they may not have even been thinking about short names at all.

Yuhong Bao
8 Dec 2008 3:06 PM
#

"On Netware 4 the default short name was simply 8.3 truncation."

This, BTW, came from HPFS. You see, LONG.NAM was originally called OS2.NAM. When Novell created the client for 32-bit Windows, they decide to just use the OS/2 namespace for their long file names.

BTW, I wonder if this was related to the fact that before the MS-IBM JDA broke up in 1991, Windows NT was called NT OS/2 and was supposed to be based on OS/2 instead of Windows?

cbloom
8 Dec 2008 3:56 PM
#

If you get into Unicode there are definitely ways that two files can have "the same" name.

1. If you're using the "A" file routines you can have multiple unicode file names which map to the same "A" code page string.

2. You can even have file names which display exactly the same *always*, by making one name with combined-accents and one with decomposed-accent characters. For all purposes these files are impossible to differentiate, except that they both exist on disk. And in fact depending on which program you use to open them, you may get one or the other.

Roger
8 Dec 2008 7:51 PM
#

I have also implemented a SMB server on Unix just like Samba. The directory routines are by far the most complex part. I ended up rewriting the code four times!

The single most difficult thing is proving a file does not exist since any of the files could map to another filename in the case insensitive or shorter name forms. And short names can change in the presence of other names.

Since the server runs in user space and there is no desire to add a database to each directory including alternate names, you pretty much have to brute force check things. Generally servers implementing Apple file sharing did end up with additional files on the server to track what was going on and be consistent.

someone else
8 Dec 2008 9:20 PM
#

What's this short file name thingy you talk about there?

Well, seriously, I usually disable 8.3 name creation ASAP and indeed tend to kick programs which don't like that, unless I *really* need them (in which case fsutil comes in handy).

It's quite amazing how many programmers haven't accepted the concept of long file names after over 15 years. Yes, Microsoft, I am looking at you.

Duke of New York
8 Dec 2008 9:43 PM
#

It's quite amazing how many people don't "get" the concept of legacy compatibility.

(CP/M? What's that?)

Terry Denham
8 Dec 2008 11:51 PM
#

Unless the person was trying to describe "foo.bar" and "foo.barbaz." (second has no extension.

copy con foo.bar

^z

copy con foo.barbaz.

^z

c:\dir /b foo.bar*

foo.bar

foo.barbaz

Larry Osterman [MSFT]
9 Dec 2008 1:05 AM
#

"BTW, I wonder if this was related to the fact that before the MS-IBM JDA broke up in 1991, Windows NT was called NT OS/2 and was supposed to be based on OS/2 instead of Windows?"

Yuhong, we've been through this before. Windows NT was never "based" on any operating system. It was (and is) a totally standalone operating system. It supported multiple "personalities" (subsystems) that allowed emulation of other operating system semantics on top of NT - operating systems like Posix and OS/2. After the IBM/MSFT breakup, the OS/2 emulation layer was scrapped (or rather was no longer the default) and replaced with the Win32 based emulation layer that is still there today.

Read Mark Russinovitch's "Windows NT Internals" and G. Paschal Zachary's "Showstopper!" for more details.

SuperKoko
9 Dec 2008 2:44 AM
#

I've seen two files with names differeing on the case only, on a CD-ROM. Windows seem to handle these files pretty well. If an exact match is passed to CreateFile, it's properly open. I don't know if it works for FindFirstFile.

If there're Foo.bar and FoO.bar

Does FindFirstFile("FoO.bar") first returns FoO.bar or Foo.bar ? Maybe it cannot return more than one... But which one?

I would expect it to find files in no particular order.

Maybe these CD-ROMs aren't conforming to standards.

On an NTFS system, there're three namespaces (POSIX, Win32 and MS-DOS). I guess it means that, for FindFirstFile/CreateFile & any Win32 function, two names will always differ by more than case.

Thomas
9 Dec 2008 7:17 AM
#

You can use a diskeditor to directly edit the directory entries on disk. So you can even have three files with the same name.

Andrew
9 Dec 2008 7:37 AM
#

That \\?\c:\temp\com2.txt trick to create a file with an otherwise restricted name (that btw cannot be deleted under Windows normally) reminded me of a huge problem I faced just recently.

My primary HDD crashed and I had saved as much data as possible using dd from a Ubuntu LiveCD to a external USB NTFS drive. On accessing the files later under Windows XP, I found that quotes/apostrophes in certain filenames had somehow been replaced by the "?" character of all things!

Now Windows as usual bugged out big-time and NOTHING I did worked to delete those files. Believe me, I tried *everything*, including downloading all sorts of tools and using esoteric commands I'd never heard of. I also spent hours with MS Tech Support and they weren't able to help me either. Now the cause may well have been a problem with Ubuntu's hacked implementation of the proprietary NTFS file system, but for the life of me I cannot understand why in blazes XP cannot even delete a file like that? IMO, if a file's there on the HDD, exists in the OS' proprietary FS and is also visible in the OS, then it darn well should also be accessible by the OS for all operations. For me, these files were just taking up space on the disk and I couldn't even touch them.

The only thing that worked finally was to boot back into Ubuntu using the LiveCD and getting rid of the offending files. Whew! Frankly, I was disappointed to say the least in MS Support and also in the devs in charge of NTFS development. Will this problem ever be attended to, possibly in Windows 7? or is that too much to ask? (No Raymond, I'm not asking you personally.)

</rant> :)

Liam
9 Dec 2008 8:51 AM
#

Unless I am missing something I do not see this behaviour, yet have two files.

G:\foobar>echo barbaz >>foo.barbaz

G:\foobar>echo bar >>foo.bar

G:\17_may_desktop\desktop_code\foobar>dir /x

Volume in drive G has no label.

Volume Serial Number is FCB5-FFD4

Directory of G:\foobar

```
09/12/2008  13:49    <DIR>                    .

09/12/2008  13:49    <DIR>                    ..

09/12/2008  13:49                 6           foo.bar

09/12/2008  13:49                 9 FOO~1.BAR    foo.barbaz

               2 File(s)             15 bytes

               2 Dir(s)  322,474,455,040 bytes free

G:\foobar>fc foo.bar foo.barbaz

Comparing files foo.bar and FOO.BARBAZ

***** foo.bar

bar

***** FOO.BARBAZ

barbaz

*****
```

Liam
9 Dec 2008 8:54 AM
#

Yes all files are in G:\17_may_desktop\desktop_code\foobar not G:\foobar I was just trying to make the entry more confined.

SuperKoko
9 Dec 2008 9:03 AM
#

"Now the cause may well have been a problem with Ubuntu's hacked implementation of the proprietary NTFS file system"

dd is a dumb byte-to-byte copy, it doesn't do such thing. If you've used NTFS-3G to recreate your partition, and used cp to copy data on it, then, the file system driver is responsible.

I guess this is a namespace issue. NTFS-3G uses the case-sensitive POSIX namespace which may include special characters (: ? etc.). Ideally, the file should be given three names, in the three (POSIX, Win32, MS-DOS) namespaces. AFAIK NTFS-3G writes in the POSIX namespace only.

I guess these files should be removed by a program using the POSIX NT API or the native NT API.

someone else
9 Dec 2008 9:32 AM
#

“It's quite amazing how many people don't "get" the concept of legacy compatibility.”

Why should 64-bit Windows try to be compatible with a 16-bit OS?

[*Because it's s compatible with a 32-bit OS which is compatible with a 16-bit OS. It's like asking, "Why should modern railroads try to be compatible with Roman chariots?" (I don't care whether it's actually true; I'm just recalling the principle.) -Raymond*]

Grant
9 Dec 2008 11:47 AM

#

Regarding Micheal's SAMBA scenario...

In that case you would have two different files with two different names. So you'll get the right file.

The original scenario is like trying to create a file foo.bar and foo.barbaz, and then making a symlink that points foo.bar to foo.barbaz. And expecting both foo.bars to exist somehow.

JamesCurran
9 Dec 2008 3:43 PM
#

ya'know, many year ago (when Win95 was still new) I too tried to enlighten the masses on such things:

http://groups.google.com/group/alt.folklore.computers/browse_thread/thread/2911b01a28124dd3#d9a0a1f7c7cf2f51

Alexandre Grigoriev
9 Dec 2008 8:07 PM
#

"Why should 64-bit Windows try to be compatible with a 16-bit OS?"

[Because it's s compatible with a 32-bit OS which is compatible with a 16-bit OS.]

16-bit programs don't run in x64 Windows anymore.

[*And Roman chariots don't run on modern train tracks. -Raymond*]

Alexandre Grigoriev
9 Dec 2008 8:11 PM
#

Andrew,

command window:

del \\?\c:\temp\com2.txt

Those files with screwed up names: double quotes are not allowed in the name. Single quotes are allowed in Win32, but supposedly Ubuntu barfs on them and converts them to a bogus character. I'm curious what the actual character was. You could paste it into an UNICODE file and then check in a hex editor.

Andrew
10 Dec 2008 12:32 AM
#

SuperKoko - "I guess these files should be removed by a program using the POSIX NT API or the native NT API."

It's Google time I guess, but I don't suppose you have any recommendations, do you? Even MS Support wasn't able to recommend any utilities that worked.

Alexandre Grigoriev - "Those files with screwed up names: double quotes are not allowed in the name. Single quotes are allowed in Win32, but supposedly Ubuntu barfs on them and converts them to a bogus character. I'm curious what the actual character was. You could paste it into an UNICODE file and then check in a hex editor."

Well, as I said, they've mercifully been deleted now. But I guess it ought to be easy enough to check. Just create a file with an apostrophe under Windows, then pop in a Ubuntu LiveCD and copy the same to an NTFS partition. (BTW, I used dd to copy from a FAT32 to an NTFS partition. Would copying from NTFS to NTFS have made any difference?) Voila! You should have an untouchable file under Windows with a ? character in place of the apostrophe.

BTW Alexandre, how would I paste the character into a Unicode file? Copy the name under Explorer and paste? Redirect the output of Dir?

SuperKoko
10 Dec 2008 7:34 AM
#

"

BTW, I used dd to copy from a FAT32 to an NTFS partition. Would copying from NTFS to NTFS have made any difference?

"

I'm not sure to understand the exact sequence of operation you performed. dd makes a dumb copy and doesn't deal with file names.

cp, tar & other file utilities deal with names.

Assuming you mount'ed the partition and called cp to copy back the data, the file name corruption happened there.

"but I don't suppose you have any recommendations, do you?"

Not really. I mean: I have never dealt with this issue. If I had to try something, I would write a small application calling NtDeleteFile.

ender
10 Dec 2008 7:42 AM
#

This reminded me of a bug I encountered in Product X that caused Windows to BSOD. When Product X created files (I was doing some screenshots), it just truncated the filename to create short names, which caused several files to have the same short name on disk. When you accessing such (corrupted) filesystem from Windows, it would cause a BSOD when you tried to open the second or 3rd file (which reminds me, I need to test if this still happens on Vista). chkdsk fixed the problem.

Steven Edwards
10 Dec 2008 9:06 AM
#

[Because it's s compatible with a 32-bit OS which is compatible with a 16-bit OS....

Raymond, this is getting a little off topic but while I can agree with this logic for things like reserved device names but for the life of me I cannot figure out why you guys carried over some of the old Win16 API that had been ported to Win32 and now exists in Win64. Case in point like WinExec and friends. I don't understand why you did not just have the compiler throw an error and force the developer to switch to a pure Win32/64 function and kill/remove the old Win16 apis that had been ported.

someone else
10 Dec 2008 11:44 AM
#

>>16-bit programs don't run in x64 Windows anymore.

>[And Roman chariots don't run on modern train tracks. -Raymond]

Isn't that a sufficient condition for "incompatible"?

[*And yet we continue to use chariot-compatible railroad gauge. -Raymond*]

Duke of New York
10 Dec 2008 1:32 PM
#

"I don't understand why you did not just have the compiler throw an error and force the developer to switch to a pure Win32/64 function and kill/remove the old Win16 apis that had been ported"

1. What the heck is this "compiler" thing you're talking about? I just want my old program to work when I run it!

2. Have fun porting a program that you can't even compile or run as-is.

Yuhong Bao
10 Dec 2008 3:06 PM
#

LarryOsterman: I know and that is beside my point. Actually I don't think it is really related.

BTW, I talked with Larry Osterman about providing a Subsystem SDK. I cited Interix as an example and pointed out that the third party that created the subsystem had to get NT source code from MS to do so, and that this should not be needed.

Ken Hagan
10 Dec 2008 4:38 PM
#

"Well, seriously, I usually disable 8.3 name creation ASAP and indeed tend to kick programs which don't like that, unless I *really* need them (in which case fsutil comes in handy)."

That's not an option open to a software developer who doesn't have administrative control over all their customers' systems.

Having said that, I'm astonished that you can actually find any such programs to kick. I first started disabling 8.3 names about 15 years ago and have never knowingly broken anything. It is (to me) astonishing that Windows still defaults to creating such names.

someone else
10 Dec 2008 8:56 PM
#

>> "Well, seriously, I usually disable 8.3 name creation ASAP and indeed tend to kick programs which don't like that, unless I *really* need them (in which case fsutil comes in handy)."

> That's not an option open to a software developer who doesn't have administrative control over all their customers' systems.

Those systems will have their own 8.3 related setting. And since I develop without 8.3 names, my programs stand an excellent chance to actually work on systems where short name creation is disabled, too.

> Having said that, I'm astonished that you can actually find any such programs to kick.

So am I. Microsoft Keyboard Layout Creator, for example. Several programs don't like spaces in filenames either (how old is Windows NT again?), but that's a different story.

Igor Levicki
11 Dec 2008 7:00 AM
#

First of all, you can have two identically named yoghurt packages in your fridge or a dozen eggs (all called "eggs"). You can also have two identical books on your bookshelf.

Going by that logic, the inability of a computer to discern two objects with the same name is completely unnatural.

The main flaw is in every filesystem design -- they are all using a name as a unique key instead of using a GUID and allowing duplicate names.

Of course, many will now dispute the usefullness of having two files with the same name and bring up the issue of chosing which file to open when the user asks the application to open a file by name. Yes, it may complicate things a bit but it is solvable.

For example if you have a file named "Raymond" which is a Word document, another file named "Raymond" which is a photo, and a third one named "Raymond" which is a video then Word should open the document, Picture Viewer should open the photo, and Windows Media Player the video. If there is an application which supports all three file types it should offer a preview dialog so the user can pick the right file.

File extensions would then be unneeded because files would be tagged by their mime type which could be specified in CreateFile() call and if not specified it would bring the preview/chooser dialog I mentioned above.

In my opinion that would make the computer usage much more natural than it is for the last 40 years or so.

When we are at filesystems, the concept of strong hierarchy should be replaced by the concept of views and tagging so that the same item can show up in multiple places/contexts.

This is one of those computing areas which have been neglected for far too long.

[*All you did was change the terminology: Under your model, the thing you pass to CreateFile is a name and type. It doesn't change the underlying question: "Can you have two files with the same name and type?" -Raymond*]

Ken Hagan
11 Dec 2008 12:33 PM
#

"The main flaw is in every filesystem design -- they are all using a name as a unique key instead of using a GUID and allowing duplicate names."

For me, the "name" is defined by "what you need to specify to uniquely identify the file". Most OS APIs take the same view (so its an OS issue, not a filesystem one). Historically, those who devised the URL spec, or the run-time libraries of most computer languages, have taken the view that this should be a string of printable characters comprehensible to the end-user. If you choose to deviate from this tradition then you may find that no-one writes apps for your OS.

Igor Levicki
12 Dec 2008 8:10 AM
#

>>It doesn't change the underlying question: "Can you have two files with the same name and type?"<<

That is not the same question you asked in your article, you have just rephrased it now.

In my opinion, you _should_ be able to have two files with the same name and type. In that case the only relevant way to determine which one to open is by the content.

>>For me, the "name" is defined by "what you need to specify to uniquely identify the file"<<

Right, but the only way to uniquely identify the file is by examining its content and here is why:

>>Historically, those who devised the URL spec, or the run-time libraries of most computer languages, have taken the view that this should be a string of printable characters comprehensible to the end-user.<<

Is that why filenames such as "xsdfgpo8df,7uyv908_34c's.jpg" are allowed?

Those are all printable characters, aren't they?

And for sure the name is unique. But is it comprehensible? Not without opening the file and examining its contents.

Furthermore, what seems to be a comprehensible string of printable characters to a Chinese person isn't comprehensible to me, not to mention that the Unicode characters (which are now allowed in file names) are not part of the printable ASCII character set.

So my question is -- what is the point of having unique file names if they still don't uniquely identify the content to the user?

For example:

- "Raymond001.jpg"

- "Raymond002.jpg"

Why is that any better and more identifying than:

- "Raymond"

- "Raymond"

?

Once you answer those questions, you will hopefully see that the current scheme is broken and those who designed it won't look that smart and authoritative anymore. After all they couldn't see this far into the future and predict all the ways we could use our computers much less predict what power will be available to do those things in a better, more intuitive way.

Rationale is that such things should not be designed by developers/programmers/coders in the future. By their numerous mistakes in UI design made so far they have shown us that they are completely incapable of user-centric intuitive problem solving.

SuperKoko
12 Dec 2008 5:58 PM
#

@Igor Levicky:

I don't think the file system API is broken at all.

Somehow, data has to be indexed by unique keys in a namespace.

The current UNIX/Windows scheme is to provide user-readable (unless you volountary make them unreadable) character strings as keys, organized in a hierarchy.

GUID could have been used as keys, yes.

In both cases, it's trivial to hide these unique keys from user's view. Just write a file explorer that shows icons & names based on the file contents, rather than on the file name/GUID.

Tagged file formats may contain things like author name, title & other useful info. With extended attributes, some file systems provide this info for every type of file. Some media or file explorer show this information, and it's easy to hide the file name.

I don't see why GUID would be better than names. IMO, names are much more in the spirit of UNIX where users are administrators and programmers. Hiding things is ok as long as you use a computer to do basic stuff, but when interoperability comes, when you've to back up or copy files between different platforms, the raw file names help.

I feel Windows Explorer is enough criticized about being opaque and hiding too many things to the user.

The other thing about current UNIX/Windows file systems is their hierarchical organization which might not be ideal, because not everything is really organized hierarchically. However, this organization is very standard and is very useful for interoperability with pretty any platform in the world.

Basically, what you want is WinFS. I think that WinFS failed (more than 15 years of development and not yet released). Maybe because the interoperability loss it too heavy. Maybe because it would require redevelopping too many tools. Maybe because the benefits are dubious.

BTW, Raymond001.jpg is not a good name but having too Raymond names is even more confusing. Our brain creates indexes too? I can remember that Raymond001.jpg is, e.g. a photo of Raymond and with his brother and Raymond002.jpg is a photo of Raymond with his father, but if both have the same name, I'll have to open both to see them... or use thumbnail preview. In that case, I don't need names anymore. Again, that's an EXPLORER issue, not a low-level OS issue.

Friday
13 Dec 2008 7:46 PM
#

Why wasn't UAC mentioned in this article?

Here, let me fix that for you.