The Old New Thing

How do I get information about the target of a symbolic link?

12 Feb 2010 7:00 AM | 26

Functions like `GetFileAttributes` and `FindFirstFile`, when asked to provide information about a symbolic link, returns information about the link itself and not the link destination. If you use the `FindFirstFile` function, you can tell that you have a symbolic link because the file attributes will have the `FILE_ATTRIBUTES_REPARSE_POINT` flag set, and the `dwReserved0` member will contain the special value `IO_REPARSE_TAG_SYMLINK`.

Okay, great, so now I know I have a symbolic link, but what if I want information about the link target? For example, I want to know the size of the link target, its last-modified time, and its name.

To do this, you open the symbolic link. The I/O manager dereferences the symbolic link and gives you a handle to the link destination. You can then call functions like `GetFileSize`, `GetFileInformationByHandleEx`, or `GetFinalPathNameByHandle` to obtain information about the symbolic link target.

Exercise: If the field is called `dwReserved0`, shouldn't it be off limits? Why isn't the field called `dwReparsePointType`?

Blog - Comment List MSDN TechNet

Comments

John
12 Feb 2010 7:06 AM
#

Well, I suppose that the file attributes can contain as yet undefined flags. When those flags are present, the dwReserved fields can contain information specific to the meaning of those flags.

Mathieu G.
12 Feb 2010 7:26 AM
#

I guess this is because this field was defined before Symbolic links existed, and its name cannot be modified now for compatibility with programs using old headers.

Am I correct ?

Leo Davidson

12 Feb 2010 8:10 AM
#

The reserved field might not always indicate the reparse point type. It only has that meaning when another flag is set.

Not that it seems worth the complexity, but it could be given a more specific name without breaking old code by using a union. On top of unions being a it icky, I guess there would be the risk some new code would set the newly-named value and also zero out the reserved field without realising they're the same. Better to leave it alone!

iddqd
12 Feb 2010 8:15 AM
#

You can also use DeviceIoControl with FSCTL_GET_REPARSE_POINT to obtain information about the symbolic link target "as is".

Daniel ZY
12 Feb 2010 8:56 AM
#

You can get the name from handle in a supported way on 2000 too (see: http://msdn.microsoft.com/en-us/library/aa366789%28VS.85%29.aspx)

640k
12 Feb 2010 9:30 AM
#

Thanks for this info. I wrote a program last week which needed to check/resolve the target of a slink!

> I guess there would be the risk some new code would set the newly-named value and also zero out the reserved field without realising they're the same. Better to leave it alone!

That's why you should use this pattern:

```
ZeroMemory(&mystruct, sizeof(mystruct));

mystruct.a=1;

mystruct.b=2;
```

Doc
12 Feb 2010 9:33 AM
#

Is this the official supported approach to resolve a symbolic link? msdn doesn't have any more info on this, this blog is now the only documented approach from msft.

> [*Nothing on this Web site is the official supported approach to anything. It's just my thoughts, not "the documented approach from Microsoft." I just found two blocks and snapped them together in an interesting way. -Raymond*]

mh
12 Feb 2010 10:06 AM
#

At a rough guess I'm going to say that it's because there were certain third-party filesystem utilities in the past that had programmers who thought "whay-hey! unused struct members to have a party in!" and now it's too late to go back.

James
12 Feb 2010 11:12 AM
#

Is there a way to find the target without opening a handle? GetFileAttributes and FindFirstFile work even if you do not have access to the file, and even if the file is opened "exclusive" by another process.

> [*I don't know. Maybe you can find a better Lego piece. -Raymond*]

Alexandre Grigoriev
12 Feb 2010 11:13 AM
#

@DanielZY,

This MapViewOfFile kludge only works until you get a handle without read-execute rights.

Teo
12 Feb 2010 11:46 AM
#

Bonus question: Why does GetFileInformationByHandleEx exist only on Vista and up, given the fact that it is a very thin wrapper around NtQueryFileInformation, which exists since NT4 (or even 3.1)?

> [*Translation: Why didn't you use your time machine? -Raymond*]

mikeb
12 Feb 2010 12:28 PM

#

@Leo Davidson:

A union might seem to be a potential solution, however adding a union to replace the name dwReserved0 would require changes in user code to add another name 'level' to that part of the WIN32_FIND_DATA structure unless you wanted people using the API to be forced to use the MSVC extension of nameless unions.

I wouldn't be surprised if there are many things in the Windows SDK that rely on MSVC extensions, but I'd guess that in general Microsoft would like to try to avoid them unless truly necessary.

Now, if there was some foresight when the structure was initially developed, the dwReserved0 member might have been a union right from the start (maybe named 'extendedinfo' or something) that initially only contained a dwReserved0 member. And they could have included the dwResrved1 member into a structure that was part of the union. Or something. But we'd need a time machine to fix that now, and honestly I can't recall a structure definition for an API that had that level of insight to make their reserved members source-level backwards (or would that be forwards?) compatible so nicely.

But now that I've mentioned it, I'll try to keep this in mind for any APIs that I might develop with reserved information - not that I develop any public-facing APIs.

Billy O'Neal
12 Feb 2010 2:45 PM
#

@mikeb:

"Nameless" are not a Microsoft extension, they are part of the standard: http://www.cplusplus.com/doc/tutorial/other_data_types/ . Scroll down to "Anonymous unions".

[*True but irrelevant. Nameless unions are nonstandard in C. -Raymond*]

Teo
12 Feb 2010 7:10 PM
#

Why didn't you use your time machine?

Urgh, no. It *works* downlevel, it just is not *supported*. You have a redistributable lib which implements it downlevel.

But the question stay, why it was not implemented earlier, given the fact that its functionality was available for ages?

Wait wait, .Net did not exist when NT4 was created, yet somehow I've seen .net programs running on NT4? Has Microsoft discovered the time machine ;-)

[*I don't know why it wasn't implemented earlier. But what's done is done, and you*

can't fix that without a time machine. -Raymond]

Gabest
12 Feb 2010 7:18 PM
#

There is a nice COM object for handling links, CoCreatable with CLSID_ShellLink.

anonymuos
12 Feb 2010 10:31 PM
#

Feature request: Please make Explorer navigate thru all types of junction points upon doubleclicking (I know that crap that Explorer is not obligated to support all NTFS features) and at least make it calculate the size correctly (exclude redundant files) so we can get the correct size of the WinSxS folder.

[*There's no point since there is no "correct" size for a folder containing hard links. -Raymond*]

Ken Hagan
14 Feb 2010 3:45 AM
#

@James:

"Is there a way to find the target without opening a handle? GetFileAttributes and FindFirstFile work even if you do not have access to the file, and even if the file is opened "exclusive" by another process."

I thought CreateFile was happy for you to request no access to the file. If so, there'd be no sharing conflict with any existing open handle or with the file's ACL.

And of course, if someone refuses you list access to the parent directory then you *shouldn't* be able to get the information. Sometimes you just aren't welcome!

unlink
14 Feb 2010 5:57 AM
#

> There is a nice COM object for handling links, CoCreatable with CLSID_ShellLink.

CLSID_ShellLink are used for .lnk-files, not file system objects (reparse points, junction points, symbolic links, or hard links).

And no, COM is not a nice technology.

640k
14 Feb 2010 3:30 PM
#

> I thought CreateFile was happy for you to request no access to the file.

Thanks for info!

Don Juan
14 Feb 2010 5:17 PM
#

Maybe Interix / POSIX for windows implements readlink() ? Simple and clean. But probably a worthless comment for general windows development.

Chris Long
15 Feb 2010 8:20 AM
#

I was just trying to use FindFirstFile/FindNextFile the other day. I was rather astonished to find that, while it very helpfully finds all files matching a path containing wildcards, it only returns the filename, not the full path to the file.

For example, asking it to find all files matching '../fred/barn*.txt' would return barney1.txt, barnes.txt, barnone.txt etc. The caller then has to parse the input path to reconstruct a useable pathname.

Not to difficult to work around but it seems odd that Windows would go to the effort of finding the full path to a file and then only return part of it.

I'll be very happy if someone points out something obvious that I missed...

[*readdir() doesn't return the path to the directory either. Both are operations which return the contents of a directory. The path to the directory itself is not part of the directory's contents. -Raymond*]

Micah Brodsky
15 Feb 2010 9:20 PM
#

Re: anonymous

As a usability feature, it actually would be nice if Explorer / shell dialogs could navigate through symlinks, to cater to us nerdy types who have "Show all files" enabled and keep hitting the symlinks rather than the real folders by accident. "Access is denied" is a dreadfully annoying error. (And while I'm wishing, my girlfriend wants a pony. ;)

[*You're not getting "access denied" because they are symlinks. You're getting "access denied" because those specific junctions explicitly deny listing directory contents. -Raymond*]

Chris Long
16 Feb 2010 8:49 AM
#

[readdir() doesn't return the path to the directory either. Both are operations which return the contents of a directory. The path to the directory itself is not part of the directory's contents. -Raymond]

Thanks for the reply, Raymond. I agree that the behaviour is reasonable, it's just less helpful than I was (reasonably, I think) expecting.

Admittedly, I'm passing in a path supplied on the command line, which is not going to be a common use of these functions. However, returning a path which can be directly used in a CreateFile call would be a very handy (possibly additional) feature.

Cheong
16 Feb 2010 5:13 PM
#

[There's no point since there is no "correct" size for a folder containing hard links. -Raymond]

Another attempt for feature request of next version of NTFS: store pointers to all location of "hard links" to the file's information block, then when counting filesize, count only if the current file information block is pointed by the first pointer. When this "first hard link" is deleted, move the pointers up one level accordingly. (Afterall, even on *nix system where both symbolic link and hard link are common, files rarely get more than 10 "links")

[*You can already do this with the FindFirstFileNameW function. But would the algorithm be useful? Why do people want to know the size of a directory? What problem does it solve? Does this algorithm help to solve that problem? (Part of my point is that there are multiple problems here, and no single algorithm solves all of them.) -Raymond*]

Craig Barkhouse (MS)
18 Feb 2010 5:27 PM
#

@Don Juan:

No need for a special API just to read the contents of a symbolic link (though the POSIX subsystem might very well provide readlink()). You can open a symbolic link or any other reparse point directly by specifying FILE_FLAG_OPEN_REPARSE_POINT to CreateFile(). Then use ordinary I/O APIs like ReadFile().

@Cheong:

NTFS already stores a complete list of all hard links to every file, and Raymond has pointed to the FindFirstFileName() API which can be used to get this list. As for file size,

there's only one file size for each file, and that's exactly what NTFS stores. Adding up file sizes for any files is not something that NTFS does and is left to an application to do however it wants. If an application wants to multiply count for every link, or singly count, or something else entirely, then it can. Similarly, if an application wants to count for alternate data streams, or not count them, then it can. NTFS makes all of the information available for whatever counting strategy is preferred. Yes it's a preference; there's no obvious one true way to do it.

Yuhong Bao
18 Feb 2010 5:33 PM
#

"You have a redistributable lib which implements it downlevel."

Actually, it is a staticly linked lib called FileExtd.lib, and disassembling it shows it uses GetProcAddress to get addresses to the NTDLL function it calls. Without a time machine, this is the best MS can do.