Naming Files, Paths, and Namespaces

306 out of 441 rated this helpful

All file systems supported by Windows use the concept of files and directories to access data stored on a disk or device. Windows developers working with the Windows APIs for file and device I/O should understand the various rules, conventions, and limitations of names for files and directories.

Data can be accessed from disks, devices, and network shares using file I/O APIs. Files and directories, along with namespaces, are part of the concept of a path, which is a string representation of where to get the data regardless if it's from a disk or a device or a network connection for a specific operation.

Some file systems, such as NTFS, support linked files and directories, which also follow file naming conventions and rules just as a regular file or directory would. For additional information, see Hard Links and Junctions and Reparse Points and File Operations.

For additional information, see the following subsections:

File and Directory Names

All file systems follow the same general naming conventions for an individual file: a base file name and an optional extension, separated by a period. However, each file system, such as NTFS, CDFS, exFAT, UDFS, FAT, and FAT32, can have specific and differing rules about the formation of the individual components in the path to a directory or file. Note that a *directory* is simply a file with a special attribute designating it as a directory, but otherwise must follow all the same naming rules as a regular file. Because the term *directory* simply refers to a special type of file as far as the file system is concerned, some reference material will use the general term *file* to encompass both concepts of directories and data files as such. Because of this, unless otherwise specified, any naming or usage rules or examples for a file should also apply to a directory. The term *path* refers to one or more directories, backslashes, and possibly a volume name. For more information, see the Paths section.

Character count limitations can also be different and can vary depending on the file system and path name prefix format used. This is further complicated by support for backward compatibility mechanisms. For example, the older MS-DOS FAT file system supports a maximum of 8 characters for the base file name and 3 characters for the extension, for a total of 12 characters including the dot separator. This is commonly known as an *8.3 file name*. The Windows FAT and NTFS file systems are not limited to 8.3 file names, because they have *long file name support*, but they still support the 8.3 version of long file names.

Naming Conventions

The following fundamental rules enable applications to create and process valid names for files and directories, regardless of the file system:

- Use a period to separate the base file name from the extension in the name of a directory or file.
- Use a backslash (\) to separate the *components* of a *path*. The backslash divides the file name from the path to it, and one directory name from another directory name in a path. You cannot use a backslash in the name for the actual file or directory because it is a reserved character that separates the names into components.
- Use a backslash as required as part of volume names, for example, the "C:\" in "C:\path\file" or the "\\server\share" in "\\server\share\path\file" for Universal Naming Convention (UNC) names. For more information about UNC names, see the Maximum Path Length Limitation section.
- Do not assume case sensitivity. For example, consider the names OSCAR, Oscar, and oscar to be the same, even though some file systems (such as a POSIX-compliant file system) may consider them as different. Note that NTFS supports POSIX semantics for case sensitivity but this is not the default behavior. For more information, see **CreateFile**.
- Volume designators (drive letters) are similarly case-insensitive. For example, "D:\" and "d:\" refer to the same volume.
- Use any character in the current code page for a name, including Unicode characters and characters in the extended character set (128–255), except for the following:
    - The following reserved characters:
        - < (less than)
        - > (greater than)
        - : (colon)
        - " (double quote)
        - / (forward slash)
        - \ (backslash)
        - | (vertical bar or pipe)
        - ? (question mark)
        - * (asterisk)
    - Integer value zero, sometimes referred to as the ASCII *NUL* character.
    - Characters whose integer representations are in the range from 1 through 31, except for alternate data streams where these characters are allowed. For more information about file streams, see File Streams.
    - Any other character that the target file system does not allow.
- Use a period as a directory *component* in a path to represent the current directory, for example ".\temp.txt". For more information, see Paths.
- Use two consecutive periods (..) as a directory *component* in a path to represent the parent of the current directory, for example "..\temp.txt". For more information, see Paths.
- Do not use the following reserved names for the name of a file:

  CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. Also avoid these names followed immediately by an extension; for example, NUL.txt is not recommended. For more information, see Namespaces.

- Do not end a file or directory name with a space or a period. Although the underlying file system may support such names, the Windows shell and user interface does not. However, it is acceptable to specify a period as the first character of a name. For example, ".temp".

Short vs. Long Names

A long file name is considered to be any file name that exceeds the short MS-DOS (also called *8.3*) style naming convention. Typically, Windows stores long file names on disk as special directory entries, which can be disabled systemwide for performance reasons depending on the particular file system. When you create a long file name, Windows may also create a short 8.3 form of the name, called the *8.3 alias*, and store it on disk also. This 8.3 aliasing can be disabled for a specified volume.

Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP: 8.3 aliasing cannot be disabled for specified volumes until Windows 7 and Windows Server 2008 R2.

On many file systems, a file name will contain a tilde (~) within each component of the name that is too long to comply with 8.3 naming rules.

Note Not all file systems follow the tilde substitution convention, and systems can be configured to disable 8.3 alias generation even if they normally support it. Therefore, do not make the assumption that the 8.3 alias already exists on-disk.

To request 8.3 file names, long file names, or the full path of a file from the system, consider the following options:

- To get the 8.3 form of a long file name, use the **GetShortPathName** function.
- To get the long file name version of a short name, use the **GetLongPathName** function.
- To get the full path to a file, use the **GetFullPathName** function.

On newer file systems, such as NTFS, exFAT, UDFS, and FAT32, Windows stores the long file names on disk in Unicode, which means that the original long file name is always preserved. This is true even if a long file name contains extended characters, regardless of the code page that is active during a disk read or write operation.

Files using long file names can be copied between NTFS file system partitions and Windows FAT file system partitions without losing any file name information. This may not be true for the older MS-DOS FAT and some types of CDFS (CD-ROM) file systems, depending on the actual file name. In this case, the short file name is substituted if possible.

Paths

The *path* to a specified file consists of one or more *components*, separated by a special character (a backslash), with each component usually being a directory name or file name, but with some notable exceptions discussed below. It is often critical to the system's interpretation of a path what the beginning, or *prefix*, of the path looks like. This prefix determines the *namespace* the path is using, and additionally what special characters are used in which position within the path, including the last character.

If a component of a path is a file name, it must be the last component.

Each component of a path will also be constrained by the maximum length specified for a particular file system. In general, these rules fall into two categories: *short* and *long*. Note that directory names are stored by the file system as a special type of file, but naming rules for files also apply to directory names. To summarize, a path is simply the string representation of the hierarchy between all of the directories that exist for a particular file or directory name.

Fully Qualified vs. Relative Paths

For Windows API functions that manipulate files, file names can often be relative to the current directory, while some APIs require a fully qualified path. A file name is relative to the current directory if it does not begin with one of the following:

- A UNC name of any format, which always start with two backslash characters ("\\"). For more information, see the next section.
- A disk designator with a backslash, for example "C:\" or "d:\".
- A single backslash, for example, "\directory" or "\file.txt". This is also referred to as an *absolute path*.

If a file name begins with only a disk designator but not the backslash after the colon, it is interpreted as a relative path to the current directory on the drive with the specified letter. Note that the current directory may or may not be the root directory depending on what it was set to during the most recent "change directory" operation on that disk. Examples of this format are as follows:

- "C:tmp.txt" refers to a file named "tmp.txt" in the current directory on drive C.
- "C:tempdir\tmp.txt" refers to a file in a subdirectory to the current directory on drive C.

A path is also said to be relative if it contains "double-dots"; that is, two periods together in one component of the path. This special specifier is used to denote the directory above the current directory, otherwise known as the "parent directory". Examples of this format are as follows:

- "..\tmp.txt" specifies a file named tmp.txt located in the parent of the current directory.
- "..\..\tmp.txt" specifies a file that is two directories above the current directory.

- "..\tempdir\tmp.txt" specifies a file named tmp.txt located in a directory named tempdir that is a peer directory to the current directory.

Relative paths can combine both example types, for example "C:..\tmp.txt". This is useful because, although the system keeps track of the current drive along with the current directory of that drive, it also keeps track of the current directories in each of the different drive letters (if your system has more than one), regardless of which drive designator is set as the current drive.

Maximum Path Length Limitation

In the Windows API (with some exceptions discussed in the following paragraphs), the maximum length for a path is **MAX_PATH**, which is defined as 260 characters. A local path is structured in the following order: drive letter, colon, backslash, name components separated by backslashes, and a terminating null character. For example, the maximum path on drive D is "D:\some 256-character path string<NUL>" where "<NUL>" represents the invisible terminating null character for the current system codepage. (The characters < > are used here for visual clarity and cannot be part of a valid path string.)

Note File I/O functions in the Windows API convert "/" to "\" as part of converting the name to an NT-style name, except when using the "\\?\" prefix as detailed in the following sections.

The Windows API has many functions that also have Unicode versions to permit an extended-length path for a maximum total path length of 32,767 characters. This type of path is composed of components separated by backslashes, each up to the value returned in the *lpMaximumComponentLength* parameter of the **GetVolumeInformation** function (this value is commonly 255 characters). To specify an extended-length path, use the "\\?\" prefix. For example, "\\?\D:\very long path".

Note The maximum path of 32,767 characters is approximate, because the "\\?\" prefix may be expanded to a longer string by the system at run time, and this expansion applies to the total length.

The "\\?\" prefix can also be used with paths constructed according to the universal naming convention (UNC). To specify such a path using UNC, use the "\\?\UNC\" prefix. For example, "\\?\UNC\server\share", where "server" is the name of the computer and "share" is the name of the shared folder. These prefixes are not used as part of the path itself. They indicate that the path should be passed to the system with minimal modification, which means that you cannot use forward slashes to represent path separators, or a period to represent the current directory, or double dots to represent the parent directory. Because you cannot use the "\\?\" prefix with a relative path, relative paths are always limited to a total of **MAX_PATH** characters.

There is no need to perform any Unicode normalization on path and file name strings for use by the Windows file I/O API functions because the file system treats path and file names as an opaque sequence of **WCHAR**s. Any normalization that your application requires should be performed with this in mind, external of any calls to related Windows file I/O API functions.

When using an API to create a directory, the specified path cannot be so long that you cannot append an 8.3 file name (that is, the directory name cannot exceed **MAX_PATH** minus 12).

The shell and the file system have different requirements. It is possible to create a path with the Windows API that the shell user interface is not able to interpret properly.

Namespaces

There are two main categories of namespace conventions used in the Windows APIs, commonly referred to as *NT namespaces* and the *Win32 namespaces*. The NT namespace was designed to be the lowest level namespace on which other subsystems and namespaces could exist, including the Win32 subsystem and, by extension, the Win32 namespaces. POSIX is another example of a subsystem in Windows that is built on top of the NT namespace. Early versions of Windows also defined several predefined, or reserved, names for certain special devices such as communications (serial and parallel) ports and the default display console as part of what is now called the NT device namespace, and are still supported in current versions of Windows for backward compatibility.

Win32 File Namespaces

The Win32 namespace prefixing and conventions are summarized in this section and the following section, with descriptions of how they are used. Note that these examples are intended for use with the Windows API functions and do not all necessarily work with Windows shell applications such as Windows Explorer. For this reason there is a wider range of possible paths than is usually available from Windows shell applications, and Windows applications that take advantage of this can be developed using these namespace conventions.

For file I/O, the "\\?\" prefix to a path string tells the Windows APIs to disable all string parsing and to send the string that follows it straight to the file system. For example, if the file system supports large paths and file names, you can exceed the **MAX_PATH** limits that are otherwise enforced by the Windows APIs. For more information about the normal maximum path limitation, see the previous section Maximum Path Length Limitation.

Because it turns off automatic expansion of the path string, the "\\?\" prefix also allows the use of ".." and "." in the path names, which can be useful if you are attempting to perform operations on a file with these otherwise reserved relative path specifiers as part of the fully qualified path.

Many but not all file I/O APIs support "\\?\"; you should look at the reference topic for each API to be sure.

Win32 Device Namespaces

The "\\.\" prefix will access the Win32 device namespace instead of the Win32 file namespace. This is how access to physical disks and volumes is accomplished directly, without going through the file system, if the API supports this type of access. You can access many devices other than disks this way (using the **CreateFile** and **DefineDosDevice** functions, for example).

For example, if you want to open the system's serial communications port 1, you can use "COM1" in the call to the **CreateFile** function. This works because COM1–COM9 are part of the reserved names in the NT namespace, although using the "\\.\" prefix will also work with these device names. By comparison, if you have a 100 port serial expansion board installed and want to open COM56, you cannot open it using "COM56" because there is no predefined NT namespace for COM56. You will need to open it using "\\.\COM56" because "\\.\" goes directly to the device namespace without attempting to locate a predefined alias.

Another example of using the Win32 device namespace is using the **CreateFile** function with "\\.\PhysicalDisk*X*" (where *X* is a valid integer value) or "\\.\CdRom*X*". This allows you to access those devices directly, bypassing the file system. This works because these device names are created by the system as these devices are enumerated, and some drivers will also create other aliases in the system. For example, the device driver that implements the name "C:\" has its own namespace that also happens to be the file system.

APIs that go through the **CreateFile** function generally work with the "\\.\" prefix because **CreateFile** is the function used to open both files and devices, depending on the parameters you use.

If you're working with Windows API functions, you should use the "\\.\" prefix to access devices only and not files.

Most APIs won't support "\\.\"; only those that are designed to work with the device namespace will recognize it. Always check the reference topic for each API to be sure.

NT Namespaces

There are also APIs that allow the use of the NT namespace convention, but the Windows Object Manager makes that unnecessary in most cases. To illustrate, it is useful to browse the Windows namespaces in the system object browser using the Windows Sysinternals WinObj tool. When you run this tool, what you see is the NT namespace beginning at the root, or "\". The subfolder called "Global??" is where the Win32 namespace resides. Named device objects reside in the NT namespace within the "Device" subdirectory. Here you may also find Serial0 and Serial1, the device objects representing the first two COM ports if present on your system. A device object representing a volume would be something like "HarddiskVolume1", although the numeric suffix may vary. The name "DR0" under subdirectory "Harddisk0" is an example of the device object representing a disk, and so on.

To make these device objects accessible by Windows applications, the device drivers create a symbolic link (symlink) in the Win32 namespace, "Global??", to their respective device objects. For example, COM0 and COM1 under the "Global??" subdirectory are simply symlinks to Serial0 and Serial1, "C:" is a symlink to HarddiskVolume1, "Physicaldrive0" is a symlink to

DR0, and so on. Without a symlink, a specified device "Xxx" will not be available to any Windows application using Win32 namespace conventions as described previously. However, a handle could be opened to that device using any APIs that support the NT namespace absolute path of the format "\Device\Xxx".

With the addition of multi-user support via Terminal Services and virtual machines, it has further become necessary to virtualize the system-wide root device within the Win32 namespace. This was accomplished by adding the symlink named "GLOBALROOT" to the Win32 namespace, which you can see in the "Global??" subdirectory of the WinObj browser tool previously discussed, and can access via the path "\\?\GLOBALROOT". This prefix ensures that the path following it looks in the true root path of the system object manager and not a session-dependent path.

Related topics

File System Functionality Comparison

Community Additions

Re: Long path support in Explorer

Adding the ability to READ (and rename) 32767 length unicode paths to explorer would be good, but allowing new paths/names to be created that exceed the MAX_PATH limit would cause more problems that it would fix. Many programs only set MAX_PATH buffers for file and path names, including much sample code.

MacBlack1
2/18/2014

Windows 8.1, still has this limitation!

I can't believe it. Windows 8.1 still has this limitation as well!! Also Windows Server 2012!

I have to use Total Commander in order to deal with deply nested files/folders. So annoying and stupid.

Please, for the love of Wozniak, revise Explorer for Windows 8.2 to support the 32,000 character file path limit that NTFS supports.

danwat12345
12/10/2013

Directory Length Limit is incorrect

Topic states

>When using an API to create a directory, the specified path cannot be so long that you cannot append an 8.3 file name (that is, the directory >name cannot exceed

MAX_PATH
minus 12).

This is incorrect since if a directory is created with a path of length MAX_PATH-12 a filename could not be added to it as a slash is needed as well as the 8.3 filename and the terminating NUL i.e.

<directory path of 248 characters>\abcdegh.abc NUL

248 +1 +12 +1 =262

dalterio
2/25/2013

mica or sassy included in the c drive

homebound is my user name which i deleted my profile trying to switch to a guest account so mny admisistration privleges would not be all over the internet.

sassybynature5150
12/29/2012

Windows Deep Directory Backup and Recovery (Quandry)

Window long file names or deep directory structures continue to be a challange for backup administrators. Without some custom integration or manual workaround, backing up and restoring deep directories on file level backups continue to be a thorn in the backup administrator's side. What word of wisdom can be offered to help lessen the impact to the user community?

Dan M 235
12/8/2012

Yet another C# library

Some time ago I created a small .NET library to solve the MAX_PATH limit for basic I/O functions.

You find it at **http://zetalongpaths.codeplex.com/** free-of-charge.

Uwe Keim

5/31/2012

MAX_PATH limitation is pretty outdated

The MAX_PATH limitation is pretty restricting for current needs. It is often needed to organize projects data into paths longer than 260 characters. I can't believe this historical piece is still being dragged behind.

Tadeusz
2/22/2012

Use 3rd Party Software

Sometimes tinkering too much about the problem will not resolve it

what I did when I encountered this issue was to download a cool tool from http://longpathtool.com/

Saved me hours of tinkering and thinking!

PhilGart
2/18/2012

A worthy alternative

Codeplex hosts AlphaFS which will enable access and manipulation of long file paths.

http://alphafs.codeplex.com/

Sh0gun
1/15/2012

Various ways around the 260 character limit

I have been working on projects where we used to have pretty long filenames decribing exactly the content *AND* pretty long folder names (describing project, version and some paramters). This was still below 260 characters. But when we moved the whole stuff to an archive-folder adding some more charcters or moved it to some user-folder on a notebook we had various unpleasant results: not copied, not being able to open, not being able to delete.

For that reason: this limit has to be removed for Win8!

Astara's
method (see below) "...However, you can 'cd' into 'C:\dir001\dir002\dir003...dir\025', ..." will not always work under
Win7 as some of the Windows shell applications such as Windows Explorer
are quite clever and recalculate the complete path length - and will deny certain
operations (e.g. rename).

There are ways to solve the various problems. The basic idea of the solutions listed below is always the same: Reduce the path-length in order to have path-lenght+name-lenth<**MAX_PATH**. You may:

- share a subfolder

- use the commandline to assign a drive letter by means of SUBST

- use AddConnectionunder VB to assign a drive letter to a path

I was able to solve all my problems using such technics.

Well, 'solve' is not all correct. Maybe I would just better use 'postpone'... Sighs...

TerraD
12/6/2011

\\?\UNC\ prefix for doesn't always work

For some operating systems prior to Windows Vista, GetLongPathName will fail when given a path that starts with the \\?\UNC\<server>\<share>\<path> form of the path, when \\<server>\<share>\<path> works with the same API. I've seen an unresolved question about GetShortPathName failing under the same scenario.

Phil Barila
10/27/2011

corrections and answer to user feedback

I wanted to make some corrections and answer a few questions. Note, I don't work for MS and probably know more about linux than MS, but I know a few things. To test many of these corrections/answers, I used the 'bash' shell from the cygwin distribution (from cygwin.com). It allows you to get around cmd.exe restrictions, but cannot, obviously, do things that the OS doesn't support.

Note that while bash can create Windows incompatible file names (like ones with '*, ?, |, >, etc...'), those that are really NT incompatible will be 'faked' and only display correctly in cygwin (it substitutes a character from the private-code page area in the Unicode character set as a place holder. It will work as the real character in cygwin-based programs, but ISN'T win-compatible. I won't bother to address characters that aren't "REAL" -- i.e. they will display correctly under Windows programs).

1), David Jameson regarding UTF-8 said Linux used a 'worse way' of filenames that were uninterpreted bytes due to low manpower in early days. Note that NT also uses this same method of storing filenames. Windows interprets those byte strings as unicode characters. It is obvious that linux can't be worse, as they both store bytes strings. Second, it wasn't due to low manpower, but was by 'design', as that was the way it was done in unix, invented back in the 70's before Unicode was around. So the choice had nothing to do with manpower nor is it any worse than NT. The current linux shell, bash, supports UTF-8 input & output, as do most desktop programs. Some old-unix compatible progs only support 1-byte characters and aren't UTF-8 compat (someday!).

For those hat asked about length on pathnames, as the base article says,'\\.\' can work some places. Another workaround that isn't suitable for general application, but can allow you create and access pathnames > 260 chars is to change directories into one of the directories in the middle of the path, then use the remainder of the path to access the filename: Example. Lets say, we have a path "C:\dir001\dir002\dir003...\dir050\myfile", where "..." are all the directories numbered between 3 and 50. That would be a total of 50 directories at 6 characters each + the separators, + 2 bytes for C:, and 6 bytes for the filename., or (50*7)+2+6->358. Way past the 260 byte limit.

However, you can 'cd' into "C:\dir001\dir002\dir003...dir\025", (where ... - dirs from 004->dir024). That will "eat up" 25 dirs,, with the drive, equaling 156 characters. The remaining path to your filename then would be referenced by "dir0026\dir0027\...\dir050\myfile", and only need a path of 160 additional characters. Neither - that way neither path will will hit the 260 char limit and you can access your file. Use this if you have an 'emergency' and find a file 'stranded' beyond the 260 byte limit -- don't use it to store files, since accessing them will be a pain. I wouldn't *expect* programs to support such path lengths or use such tricks to access them, so you might be causing yourself much pain -- but it's good to know about the trick if you need to access a file that somehow got created with too long of a pathname.

Last point: About spaces and dots in filenames and directories. The limits are in the windows shell -- not in Windows or NT. Using 'bash', you can create files with spaces (or dots), both, at the beginning and end of a filename. You can then list and open those files in explorer, and you can 'list' them in the shell (cmd.exe), but you won't necessarily be able to open them from the shell (especially trailing spaces and dots). The shell strips them out. If you want such things, use another shell, but WARNING -- if you put spaces after your filename, it will be hard to figure out how many there are. To do it, you'd have to dump your directory listing to a file and do a hexdump of the file to see how many space characters there actually are. Not a major problem, but still a bit of a pain.

I am going to have to experiment with the \\?\ prefix, as I don't know how many programs will support it -- but if a program has problems with the pathlength and doesn't support \\?\, then I'd suggest filing a bug report or an 'RFE' (Request for Enhancement).

hopefully that answers a few of the questions here -- and NOTE: I mention this again. I represent MS in no way and have no official job related to MS, so you can take what I say at face value, but not as any 'official statement'. Cheers!

Astara

p.s. - For DISPLAY purposes, (i.e. for how they look, not functionally), you can display all of the reserved characters in a filename if you use their equivalent in the unicode area "Halfwidth and Fullwidth Forms". Depending on the font you use in Explorer (or whatever program displays your filenames) they may not look exactly the same, but are usually suitable. I often use "：", or "FULLWIDTH COLON" (U+FF1A) in titles of songs where a colon should go. The characters:
  : * < > / \ | ?

all have fullwidth equivalents:
    ：＊＜＞／＼｜？

Note though, that those characters are only for display -- and won't function the same as their ASCII equivalents. They are also *dependant* on having a character set loaded that will display them (with some character sets you may get a blank or some generic symbol like a square). *Caveat 'inhibeum' (User Beware).*

BJØRN78
8/14/2011

Length Limitation

The length limitation in the API is starting to cause problems. It is like back with DOS where we had to stuff applications into 640k of memory and hack with the system and autoexe.bat files.

Why can't the OS now handle paths greater than 260 uniformly for all applications and not this current hodge-podge of support. If you find yourself with a path to long and can't delete the Microsoft solution is to share the path and then access the shared folder to delete sub folders. Repeat this process for deeper paths.

I think it is time that the technical details of why are handled better and provide the user a better experience with the file system.

BJØRN78
8/14/2011

File path limit is limiting

When can we get beyond the file path limit of 260?

640 KB barrier: gone
8.3 file limit: gone

file path limit... ugh

.MARK
6/23/2011

Non-ascii (128-255) bytes NOT passed through unchanged in ANSI file functions

When accessing files using the single-byte character functions (fopen, _open, _stat, _utime, CreateFileA, etc.), you should bear in mind that these functions all modify the name using the current codepage (what the user has set in their regional setting "language for non-unicode programs"). While this doesn't cause any problems when using an English codepage (you can use chars 128-255 in filenames), if the user is using Chinese codepage then Windows will assume that filenames passed to the ANSI functions are in gb2312 format, so therefore you cannot simply pass in any characters you want in the filename and expect it to work. This may cause problems if you are writing utf-8 encoded filenames - it might *seem* to work fine when testing using the English codepage, but if you change to Chinese it will break. The only solution is to always use the wide character versions of the functions (_wfopen, etc.)

In actuality, paths are considered to be encoded in the same "ANSI" codepage as everything else; therefore, you can only sensibly pass utf-8 encoded filenames to the ANSI function when that codepage is itself UTF-8 (CP 65001). Filenames in NT-based Windows systems (which includes everything newer than ME) are all treated as Unicode internally, even on FAT filesystems. (Which is a really good idea, which I think they stole from Plan9, though Plan9 used UTF-8 for it's text rather than UTF-16, and didn't support locale-specific encodings; it was explicitly not trying to deal with a bunch of legacy data and programs. Linux took the "worse" way, where filenames are treated as uninterpreted bytes, because of "worse is better" / low manpower in the early days, and we're still suffering a bit from that.

Samuel Bronson
1/29/2011

typo in this article?

Hi

I believe that there is a typo in this article.

"\\.\PhysicalDisk*X*"

should possibly be

"\\.\PhysicalDrive*X*"

Jozsef

Jozsef Bekes

9/3/2010

.NET support for long paths

The .NET Framework does not currently (from 1.0 to 4.0) support long paths, however, we have published an experimental library over on CodePlex that provides classes to allow you to use long paths in your projects:

Long Path

http://bcl.codeplex.com/wikipage?title=Long%20Path&referringTitle=Documentation

seiko1968
8/28/2010

Extending character limit of path filename

I have a folder tree with many files and subfolders which have know reached their character limit of 260. It is an NTFS file system and I wis h to continue adding files and subfolders. How do I extend the character capacity? I tried adding \\?
\ as a prefix as this article mentions but this was of no help.

Appreciate the feedback.

TheUniversalOne
6/22/2010

Using direct raw file system access without parsing strings

Ulf,

using periods at the end and at the beginning of file and folder names is supported by Win32 file I/O. The fact that you cannot create files or folders starting or ending with period only indicates that the shell itself prohibits you from doing that. If necessary, you can create suc h files using FAR Manager (just as an example of how you can do without playing with CreateFile method yourself). If necessary, you can create a folder that will have three dots in its name. However, you cannot create a folder with two dots (..)as this is handled as a parent dir ectory.

```
D:\Tmp>dir

 Volume in drive D is Data

 Volume Serial Number is 3063-35CE


 Directory of D:\Tmp


20.04.2010  19:48    <DIR>          .

20.04.2010  19:48    <DIR>          ..

20.04.2010  19:46    <DIR>          ...

20.04.2010  19:14               8 .tmp

20.04.2010  19:20               6 tmp.
```

```
2 File(s)          14 bytes

3 Dir(s)  128 056 749 056 bytes free
```

Exotic Hadron
4/20/2010

About periods in names of a directories

Under "Naming conventions" above it says

Do not end a file or directory name with a space or a period. Although the underlying file system may support such names, the Windows shell and user interface does not. However, it is acceptable to specify a period as the first character of a name. For example, ".temp".

However ".temp" as a folder name doesn't seem to work (at least) in Windows 7?

An other issue: In http://support.microsoft.com/default.aspx?scid=kb;en-us;905231
it says that "You cannot use the period character consecutively in the middle of a folder name.", but directory names like "start..end" *seems* to work, at least in Windows 7?

I would like to know what actully is supported and what is just a bad idea to use..

/Ulf L

Ulf L
3/5/2010

File component length

In a previous iteration of this document, it is mentioned that

```
The Unicode versions of several functions permit a maximum path length of
approximately 32,000 characters composed of components up to 255 characters in
length.
```

This information is now gone. You should add something to the "Basic Naming Conventions" that details this little fact; it gets very confusing when things fail and this bit of information is nowhere to be found. Even in the old docs, this piece of information was easy to miss.

[MSFT - Mark Amos] This information is in the section titled "Maximum Path Length".

Mark Amos - MSFT
7/30/2009

Is it possible to create files with character < > : " / \ | ? *

Is there any possibility that some application can create a file which contains the illegal characters e.g < > : " / \ | ? *

If the answer is yes, what is the behavior of GetFileAttributesEx to such files?

basantk
7/24/2009

Vista explorer removes leading spaces

Explorer of Windows Vista and Windows 7 RC removes leading spaces on file- and directory names when copying. If you happen to have a file "example.txt" and a " example.txt" (with leading space) in the same source directory, it will even try to write one over the other at the target directory of the copy operation.

Explorer of Windows XP preserves leading spaces on file- and directory names when copying.

marble65
6/26/2009

Unicode paths - NFC or NFD?

Is the Unicode UNC long path file name ("\\?\" starting ones) supposed to be normalized as NFC or NFD (or NFKC or NFKD)? (I put NFKC and NFKD in parenthesis because I presume those normalizations are not suitable. But I may be mistaken.)

Are there recommended Microsoft Win APIs which perform UNC long path file name normalization? Or are we supposed to roll our own?

Eljay451
6/24/2009

Removing files with illegal names

Although it is not recommended to name a file using reserved words like CON, sometimes they end up on disk through any number of ways. To remove them, try this method from a command prompt with appropriate priviledges. Let's say you have this offending file in the root of the C: drive. Here are the two commands to try:

```
C:\> rename \\.\C:\CON. deleteme
C:\> del deleteme
```

Mark Amos - MSFT
10/31/2008