

Long File Names LFNDir

Rick Knoblauch

Introduction

An Antidote to Aliases, LFNDir displays long filenames under DOS, instead of cryptic aliases containing tildes. After years of our enduring the cryptic constraints of DOS-style filenames, the long-filename support introduced with Windows 95 is a welcome relief. You can now name a report First-Quarter Earnings.doc instead of 1STQTRER.DOC. But if you boot from a start-up disk or restart your system in DOS mode, the descriptive long filenames are replaced by odd-looking "aliases" that conform to DOS filename constraints. Your report file will appear as FIRSQ~1.DOC.

This occurs because Windows 95, with its supporting cast of virtual device drivers, isn't loaded. Without these drivers, particularly the installable file system manager (IFSMgr), which contains the heart of the system's long-filename support, long filenames appear not to exist. They're still on the disk, but DOS knows nothing about them and ignores their existence.

Ordinarily, you'll have no reason to boot to DOS and look at your hard disk. But it may be the only way to access and retrieve files from a hard disk that's going south. There are few things more frustrating than being unable to see long filenames when you urgently need to make copies of critical files. This issue's utility, LFNDir, is a command line program that lets you view long filenames under DOS. The syntax and output for LFNDir closely match those of the DOS DIR command when DIR is executed from within a Windows 95 DOS box. You'll still need to use the alias to copy or delete a file, but at least you'll know which alias corresponds to the file you're after!

For Programmers Only

LFNDir is a 16-bit command line program that lets you view Windows 95 long filenames under DOS. The source code for LFNDir, which was written in C, is also provided for those interested in seeing how it works. LFNDir can be used under DOS or within a Windows 95 DOS box. Also, if a removable disk is written under Windows 95 and then taken to a DOS platform, LFNDir can assist in deciphering the disk contents.

LFNDir supports all DOS disk formats. The new FAT32 disk format introduced with Windows 95B (OEM Service Release 2) is not supported. LFNDir's source code demonstrates several interesting programming techniques, including how to determine the layout of a disk; directly access directory and FAT sectors; and retrieve long filenames.

Using LFNDir

To install the program, copy the file LFNDir.exe to a directory in your path (for example, the Windows Command directory). The syntax for LFNDir is shown in Figure 1. If LFNDir's options seem familiar, it's because they closely mirror those used with the DOS DIR command. The only relevant DIR options not implemented by LFNDir are /S (recurse subdirectories) and /O (sort). I decided to omit both of these options to make things simpler.

As with the DIR command, if you simply type LFNDir on the command line with no parameters, the utility will list the files in the current directory. Alternatively, you can specify a drive letter, a path, or a filename (long or DOS-style). If you use a long filename with embedded spaces, you must place it in double quotes--for example, LFNDir "file with spaces in name.txt." LFNDir supports the wildcards * and ?. As with DIR, you can enter only one file specification on the command line.

Long File Names LFNDir

Rick Knoblauch

The LFNDir command syntax is displayed when you use the /? option.

```
LFNDir [drive:][path][filename] [/P]
[/A[:attributes]] [/B] [/L]
```

where:

```
[drive:][path][filename]
```

Specifies the drive, directory, and/or files to list.

/P Pauses after each screenful of information.

/A Displays files with specified attributes:

D Directories R Read-only files

H Hidden files A Files ready for archiving

S System files - Prefix meaning not

/B Uses bare format (no heading information or summary.)

/L Uses lowercase for displaying aliases.

LFNDir improves on the DIR command by allowing you to use wildcard characters in specifying pathnames. This can save a lot of typing. For example, to list the files in the C:\Program Files\Accessories\HyperTerminal directory, you can use this command:

```
LFNDir c:\pro*\acc*yp*\*.*
```

You can use wildcards right up to the last path element. At the end of the file specification, the wildcards are interpreted just as they are with the DIR command. Some examples will help clarify. If you have a directory called Programs and you enter the parameter c:\pro*, the directory entry itself will be listed, not the files within it. To see the files in the directory, you'd enter the parameter pro**.*. If the directory name contains an extension, you must supply the actual extension or a wildcard. For example, if you're looking for files in the directory Programs.pcm, you would need to enter the parameter pro*.*.* or pro*.pcm*.*.

LFNDir will replace each wildcard directory name with the first matching directory that it finds. Since this may not be the one you want, be sure to type a sufficient portion of the directory name before using the asterisk.

Tips and Tricks

LFNDir lets you view long filenames under DOS, but to copy or delete files you must use the native DOS commands. Still, LFNDir can help. Specifying aliases for a large number of files can be made much easier by using LFNDir's output to create a batch file. Execute LFNDir in the usual way, but instead of viewing its output on the screen, redirect it to a file with an extension of .bat. For example:

Long File Names LFNDir

Rick Knoblauch

```
LFNDir c:*. * > docopy.bat
```

Next, read the file into your favorite text editor. Using the descriptive long filenames included in the listing, identify the files that you wish to copy. Delete the entries in the listing for the files you won't be copying, and also delete all of the long filenames and time/date and size information, leaving only the aliases for the desired files. Then, using your editor's copy and paste commands, insert the DOS COPY command in front of the alias for each file you wish to copy, and insert a destination for the COPY command at the end of each line. Save the batch file to disk. When you execute the batch file, the files will be copied. Note that the long filenames will not be copied, since the DOS COPY command is not aware of them.

Under normal circumstances, it's a simple task to obtain the names of all the files that match a given file specification. You pass the file specification to the relevant DOS functions, and DOS takes care of the rest. It transparently navigates the directories and FAT (file allocation table) as it locates each directory in the specified path, and then it identifies the matching files. LFNDir has to do all this without the benefit of those handy DOS functions, and not surprisingly, this takes a large amount of work.

The first step is to determine the layout of the disk, including the location of the first root-directory sector, the number of FAT sectors, and the sector where the first cluster begins. LFNDir then reads the root-directory entries, seeking the one that defines the first directory in the desired path.

When this directory entry is found, LFNDir extracts the starting-cluster number and begins reading the directory entries, seeking a match for the second directory on the desired path. If it gets through the first cluster without a match, it consults the FAT to locate the next cluster belonging to the current subdirectory. When it finds the directory entry corresponding to the next directory on the desired path, it repeats the process.

Eventually LFNDir locates the starting cluster of the last directory in the desired path. At this point, it reads all the directory entries as before, but this time it seeks matches for the desired file specification. For each found file specification, it extracts both the DOS-style filename and, if present, the long filename.

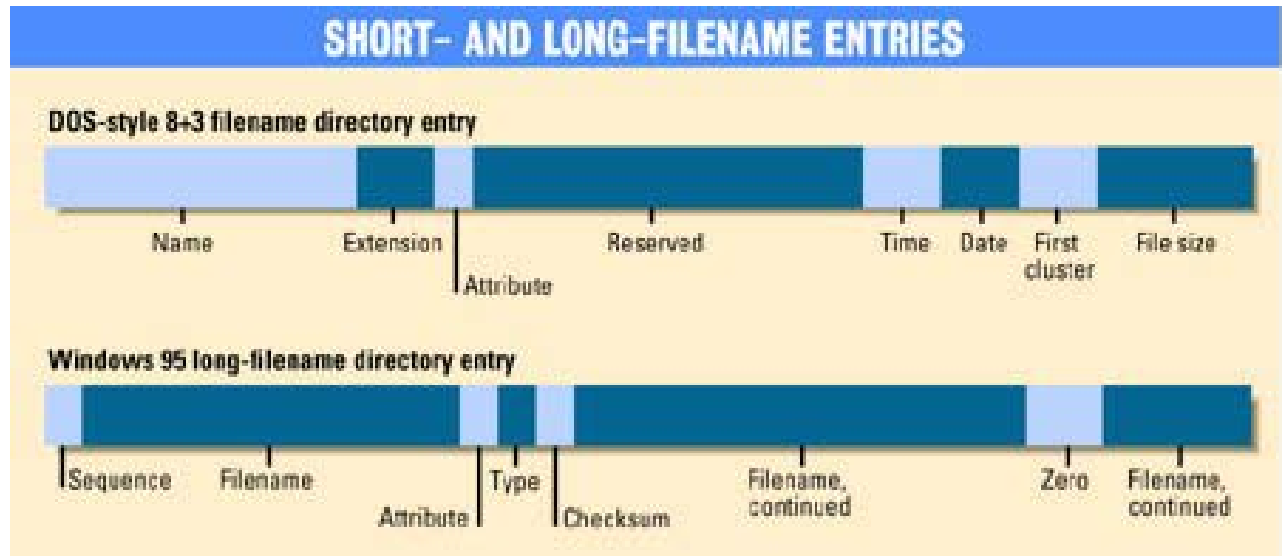
Since LFNDir must access disk sectors directly, it must use its own intimate knowledge of directory structures and the FAT to find its way to the desired files. Let's take a look at some of these file system structures before delving too far inside LFNDir.

The FAT Structure

Under the 12-bit and 16-bit FAT file systems, each disk has a fixed number of root-directory entries. (The FAT32 file system doesn't have this limitation.) These are located in the sectors that immediately follow the FAT sectors. Typically, there are two copies of the FAT stored here to provide an extra measure of protection against data loss. Each 512-byte sector in the root directory can hold 16 32-byte directory entries. Each directory entry contains the filename, file extension, attribute byte, and other basic information (see Figure 2).

Long File Names LFNDir

Rick Knoblauch



Subdirectories also use this format, but the number of subdirectory sectors is not fixed. Space is allocated dynamically for subdirectories, just as it is for data files. When a subdirectory is created, a corresponding entry in the parent directory is generated with an attribute-byte value that differentiates the subdirectory from data files.

The starting-cluster number in the DOS directory-entry structure is an index into the FAT, and it indicates where on the disk the file begins. Each entry in the FAT, which is simply a linked list of allocated data, represents a cluster of data on the disk. One cluster is the smallest allocation unit used by the system. Even if a file contains only 1 byte, it will consume an entire cluster on the disk. Clusters are composed of 1 to 64 sectors, depending on the disk size, the smaller the disk, the smaller the number of sectors per cluster. The number of sectors per cluster is always a power of 2.

Each in-use entry in the FAT either contains a link to the next cluster allocated for a given file, or indicates that the allocation chain for the file ends with that particular cluster. Depending on the disk size, FAT entries occupy either 12 bits or 16 bits. Disks with fewer than 4,087 clusters use FAT12; otherwise, disks use FAT16. The new FAT32 file system uses 32-bit entries and can support disks as large as 2 terabytes. Under FAT12 and FAT16, the FAT is always located in the sectors that immediately follow the disk's boot record. Under FAT32, the FAT location isn't fixed.

The layout of the long-filename directory entries is also shown in Figure 2. The attribute byte, which is in the same place in both structures, enables long filenames to coexist with DOS directory structures. Windows 95 places long filenames within contiguous 32-byte directory entries, using as many as necessary (and sometimes actually spanning directory sectors) to represent filenames of up to 255 characters plus the null terminator. It flags these with an attribute-byte value of 0xf, which represents an impossible combination of read-only, hidden, system, and volume label. The DOS FAT file system ignores such items. This allows long filenames to live in harmony with the existing operating-system components.

There are three filename-storage areas comprising 26 bytes, but this space can accommodate only 13 characters. This is because long filenames are stored in Unicode format, which requires 2 bytes of

Long File Names LFNDir

Rick Knoblauch

storage per character. The checksum and sequence number fields are used by Windows 95 to verify the integrity of the long filename.

Long filenames differ from DOS-style filenames in other ways besides their length. They also differ in the characters they are allowed to contain. Long filenames can contain multiple periods, embedded spaces, and other characters, such as the plus sign, comma, semicolon, equal sign, and left and right square brackets, that were previously prohibited. (Actually, spaces in filenames were permitted in later versions of DOS, but most programs, including DOS utilities, didn't support this). Also, case is preserved in long filenames, so that both uppercase and lowercase characters can be used.

We've seen how a long filename can coexist with the DOS directory structures, but that doesn't explain how DOS and Windows programs written prior to the advent of long filenames can access these files. As alluded to earlier, for each long filename, Windows 95 generates a DOS-style filename known as an alias. The directory entry for a long filename's alias immediately follows the entries for the long filename.

Windows 95 attempts to create the alias using the first six characters of the long filename followed by a numeric tail. The tail consists of the tilde character followed by a number that differentiates the alias from any others that share the same first six characters. If the long filename contains an extension, the first three characters of the extension are used as the extension for the alias. The first six characters of the alias do not match the long filename character-for-character if the long filename contains leading periods or embedded spaces, since these are skipped when Windows derives the first portion of the alias.

This summary only touches the surface. For more information on long-filename rules, see the book by Adrian King called *Inside Windows 95* (Microsoft Press, 1994) or the article on the Microsoft Developer's Network CD by Walter Oney called "Unconstrained Filenames on the PC! Introducing Chicago's Protected Mode FAT File System" (Microsoft Systems Journal, August 1994). With this background information behind us, let's look further into the details of locating and retrieving long filenames.

Locating the FAT

Since LFNDir will be going directly to the disk's FAT and directory areas, it must determine which sectors are inhabited by these DOS file system structures. This is readily accomplished by issuing the INT 21h Get Device Parameters function. This call returns a structure that describes the layout of the disk, including the number of FATs, number of sectors per FAT, and number of root-directory entries.

It's easy to find the location of the first root-directory sector. Just multiply the number of FATs by the sectors per FAT, and add the result to the location of the first FAT sector. The first FAT sector immediately follows the boot record, which is the first logical sector of the disk. Finding where subdirectories start isn't as straightforward, since these are not stored in a fixed location. I'll discuss this a little later.

Later in the search process, we'll need to read the contents of subdirectories from the disk. To do so, we'll have to translate the FAT entries for the starting and subsequent clusters into actual sector locations on-disk. Each FAT entry is an index into the disk's file area, the area of the disk that can be mapped by the FAT. So to prepare for this task, we calculate and save the sector number where the file area begins. This location immediately follows the root-directory sectors, and it is calculated by

Long File Names LFNDir

Rick Knoblauch

adding the number of root-directory sectors to the location of the first root-directory sector. Armed with the location of the root-directory sectors, the number of FAT sectors, and the first sector of the file area, we're now prepared to start searching for the desired files.

Searching the Directories

There are several LFNDir functions involved in the search process. At the highest level, `do_dir()` moves through each directory of the path and dispatches to the functions `find_in_root()` and `find_in_sub()`, which retrieve and search the appropriate directory sectors.

In order to decide how many calls to make into these functions, `do_dir()` must determine how many directories are involved in the path portion of the file specification. It starts by counting the number of backslash characters in the file specification. This count will not necessarily equal the final number of levels to be searched. The number of levels is never known for certain until the last portion of the path is found. For example, if the file specification passed to LFNDir is `C:\Level1*.*`, there are two levels: the root directory and the Level1 directory. But if the file specification is `C:\Level1\Foo` and if `Foo` turns out to be a directory, there are three levels and the search is treated as `"C:\Level1\Foo*.*"`. So when `do_dir()` is about to process the last path element, it calls `detect_subdir()` to determine if the last element is a subdirectory. If so, it tacks on the `*.*`, as shown in the example.

When locating the first directory of the path, `do_dir()` calls `find_in_root()` to search the root directory. To find all other levels within the path, `do_dir()` calls the `find_in_sub()` function. The `do_dir()` function passes these functions a pointer to the desired directory or filename, the attribute portion of the search criteria, and a pointer to a `find_context` structure that dictates where the search should start. The `find_context` structure stores the location of the last matching file or directory. The location information includes the directory sector, the entry offset within the sector, the last cluster, and the sector offset within the cluster.

The `find_in_root()` function loops through each root-directory sector calling LFNDir's `find_match()` function, which performs all of the detailed matching logic. The `find_in_sub()` function also calls `find_match()`, but instead of simply looping through a fixed number of directory sectors, it must walk the FAT chain of clusters allocated for a given subdirectory. To accomplish this, it starts with the cluster number found in the directory entry for the subdirectory. It locates the first sector of the cluster and searches each sector within the cluster. After a cluster's worth of sectors has been inspected, it extracts from the FAT the next cluster in the chain. This continues until it detects the end of the allocation chain for the subdirectory. The `find_match()` function called by `find_in_root()` must jump through quite a few hoops to find matching files and retrieve their long filenames. Let's take a look at these final steps leading to the long filenames we're seeking.

Matching the Files

The `find_match()` function searches all entries within a given directory sector. It may start at the beginning of the sector or, if it has found previous matches, resume searching within the sector. The function returns when a match is found or when the end of the specified directory sector is reached. During its processing, `find_match()` carries out a number of tasks. It matches the filenames, matches the attributes, retrieves the long filenames, and maintains a count of the number of directories or filenames it has encountered. Some of the logic required is surprisingly difficult.

First, `find_match()` must determine whether the path element it's searching for is a DOS-style filename. (This isn't totally straightforward, because names short enough to be DOS-style may

Long File Names LFNDir

Rick Knoblaugh

contain characters such as embedded spaces that make them "long" filenames.) Next, picking up where it left off on the last match, `find_match()` must go through the directory entries in search of an alias (recall that long filenames precede their corresponding aliases). If the filename is DOS-style, it can compare the name to the alias; otherwise it must compare the name to the long filename.

In either case, prior to matching the filenames, `find_match()` must match the attribute byte in the directory entry for the alias with the search attribute. Users can override the default value of normal/read-only with the `/A` switch. If the values don't match, `find_match()` skips the entry and inspects the next one. After comparing the attribute byte, it compares the filenames.

When wildcards are used, the file specification must be checked against both the alias and the long filename, even if the file specification is short enough to be a DOS-style filename. For example, consider the long filename "this123456789," whose alias is "this12~1." The file specification "this123*" is short enough to be a DOS-style filename, but it will not match the alias. Therefore, if the file specification contains a wildcard and there's no match to the alias, the long filename must be checked as well.

The `find_match()` function then calls another LFNDir function, `retrieve_lfn()`, to acquire the long filename associated with an alias. The directory entries that hold a long filename always immediately precede the entry for the file's alias, so this function examines the entries just before the alias to pull out the long filename.

The set of entries that define a single long filename can actually span directory sectors, so if `retrieve_lfn()` reaches the beginning of the sector and the long filename still has not been fully extracted, it must search the previous directory sector as well, starting at the end. As it reads each directory sector, it saves the previous sectors' contents in holding areas (`prev_root_buf` for root-directory sectors and `prev_sub_buf` for subdirectory sectors). A pointer to the appropriate previous-sector buffer is stored in the `find_context` structure to facilitate long-filename retrieval.

If `find_match()` finds a match, it finishes up its work by adding to some counters. You may have noticed that the DIR command displays the total number of directories and files that were listed. To enable LFNDir to do this as well, `find_match()` looks at the attribute byte for the directory entry of the alias and, depending on its value, adds to the `find_context` structure's `number_dirs` or `number_files` elements before returning. The search functions then return to `do_dir()`, which wraps up all this activity by finally yielding what we've been seeking from the start: a display of the desired DOS-style filename and its long filename counterpart.

With operating-system features, as well as with life in general, it's difficult to go backwards once you've grown accustomed to a new level of fulfillment. This is certainly the case with long filenames; it's painful when they disappear. The next time you must restart your system in DOS mode or find yourself working on a DOS platform with a disk volume that's been written under Windows 95, don't suffer with those DOS-style aliases. Launch LFNDir and unveil those wonderful long filenames.

Screenshot

LFNDir is a command line program that lets you view long filenames under DOS. Its syntax is almost identical to the DOS DIR command, and its output looks like what you get when you issue a DIR command in a Windows 95 DOS box.

Long File Names LFNDir

Rick Knoblauch

```
MS-DOS Prompt
8 x 16
C:\Program Files>lfnDir

Volume in drive C is RICKS
Volume Serial Number is 3441-1201
Directory of C:\Program Files

.                <DIR>          05-28-96  8:00p  .
..               <DIR>          05-28-96  8:00p  ..
PLUS!            <DIR>          05-28-96  8:01p  Plus!
ACCESS-1        <DIR>          05-28-96  8:00p  Accessories
THEMIC-1        <DIR>          05-28-96  8:01p  The Microsoft Network
MICROS-1        <DIR>          05-28-96  8:01p  Microsoft Exchange
COMMON-1        <DIR>          01-03-97  10:00a  Common Files
MICROS-2        <DIR>          01-16-97  5:56p  Microsoft Developer Network
SYMANTEC        <DIR>          07-11-97  10:14a  Symantec
0 file(s)       0 bytes
9 dir(s)        218,693,632 bytes free

C:\Program Files>_
```