UMSFUSE

A FUSE driver and tools for UMSDOS

Mike Swanson

This manual documents UMSFUSE version 1.1.1 [6c8abf6aca] 2026-11-12 23:29:00, a FUSE implementation of UMSDOS and its companion tools.

Copyright © 2026 Mike Swanson

Redistribution and use in source (Texinfo) and "compiled" forms (HTML, PDF and so forth) with or without modification, are permitted provided the following conditions are met:

1. Redistributions of source code (Texinfo) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.
2. Redistributions in compiled form (converted to HTML, PDF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This documentation is provided by the UMSFUSE project "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the UMSFUSE project be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this documentation, even if advised of the possibility of such damage.

Table of Contents

1 Introduction ... 1

1.1 What is UMSDOS ... 1
1.2 A brief primer on FAT ... 1
1.3 Why UMSFUSE ... 2
1.4 Compatibility goals ... 2
1.5 What UMSFUSE provides ... 3

2 How it works ... 4

2.1 Host tree and the EMD ... 4
2.2 Unix names and mangling ... 4
2.3 Modes, owners and types ... 4
2.4 Hard links ... 4
2.4.1 Old and new formats ... 5
2.5 Symlinks ... 5
2.6 Timestamps ... 5
2.7 Overmounting ... 6
2.8 Changes made directly to the host files ... 6
2.9 Directory cache ... 6
2.9.1 Memory ... 7
2.9.2 Writes ... 7
2.9.3 Changes on the host while mounted ... 7
2.10 Hard-link index ... 8
2.10.1 Contents ... 8
2.10.2 A hint only ... 8
2.10.3 Same results ... 9
2.10.4 Memory ... 9

3 UMSFUSE programs ... 10

3.1 `umsdosfs` ... 10
3.1.1 Deviations from the kernel ... 11
3.1.2 Hard link formats ... 11
3.1.3 Limitations from UMSDOS ... 11
3.2 `umsfsck` ... 12
3.3 `umscat` ... 15
3.4 `umscp` ... 15
3.5 `umsls` ... 15
3.6 `umsstat` ... 18
3.7 `umstar` ... 19
3.7.1 Interoperability ... 21
3.8 Common options ... 21

4 UMSDOS specification ... 22

4.1 Directory layout ... 22

4.2 Record format 23
4.3 Name fields and mangling 24
4.4 Metadata fields 26
4.5 Timestamps in detail 27
4.6 Hard-link records 28
4.7 Special files 29
4.8 Foreign layouts 30
4.9 Invariants 31

Appendix A Glossary 32

1 Introduction

1.1 What is UMSDOS

UMSDOS ("Unix" + "MS-DOS") originated in 1992, publicly released as patches for Linux 1.0.x in January 1994, and became a built-in driver beginning with Linux 1.1.36, where it remained a supported file system through the 2.4.x series.

It extends the FAT file system (in Linux, known as the "msdos" file system) with support for long file names and unrestricted character sets, file modes, ownership, and symbolic and hard links. All the while it maintains compatibility with the FAT file system and MS-DOS by never modifying nor changing the low-level file system structure. It accomplishes this goal by using "Extended MS-DOS Directory" (EMD) files that are stored on FAT with the `--linux-.---` name, one in each directory managed by UMSDOS.

Most usefully, this enables computers running MS-DOS, Windows 3.1, and Windows 95/98 to dual-boot with Linux without requiring a new dedicated partition, and the Slackware distribution even shipped a "ZipSlack" version, where installation involved unzipping a '`zipslack.zip`' file resulting in a fully functioning Linux install in the `C:\LINUX` directory. A DOS boot loader program named `loadlin.com` would then load Linux into memory, replacing DOS and booting Linux, whose root file system is contained within the DOS `C:\LINUX` directory.[1]

Not only did this share disk space, it also made experimenting and trying out Linux in the 1990s simple. Installation and uninstallation both could be handled by merely copying and deleting files. Changing hard disk partitions to assist a dual-boot with a native file system such as ext2 would usually require resizing the existing FAT partitions, at the time a process that required purchasing extra software (such as PartitionMagic) to achieve.

1.2 A brief primer on FAT

In the original design of FAT, and released with PC DOS 1.0 in 1981, the file system was extraordinarily simple, and some of the limitations were made to keep the metadata structures easy to implement on 8-bit and 16-bit home computers of the time.

DOS itself had modest requirements and they led directly to the design choices of FAT:

- Files only have a single timestamp attached to themselves, the modification timestamp.
- Names on FAT are stored in all-uppercase, the operating system providing case-insensitive lookup by transforming ASCII characters a–z to A–Z.
- File attributes available are archive, hidden, system, and read-only.

File names deserve some special attention to highlight how restrictive they are: they consist of a "base name" of up to eight characters and an optional "extension" of up to three characters, resulting in a scheme referred to as "8.3" or "8+3." The permitted character set is restricted to a subset of printable ASCII characters, excluding control characters, spaces, and many kinds of punctuation that are used in the DOS shell for command line and wildcard processing.

[1] This is actually similar to how Windows 3.1 and Windows 95/98 boot up: '`win.com`' replaces DOS in memory with the Windows kernel. Windows moves DOS into a special virtual machine it can restore and "exit" Windows back into the original DOS environment, whereas Linux has no such functionality.

In 1994 with the release of Windows NT 3.5, and in a much wider rollout in 1995 with the release of Windows 95, Microsoft expanded on the file system to permit long file names, and an extended character set than MS-DOS supported. This extension is often called "VFAT," derived from Windows 95's '`vfat.vxd`' file system driver. In VFAT, the file system structures are manipulated such that long names are stored next to the 8+3 compatibility name. Long entries are marked with special attribute bits such that MS-DOS and legacy applications would ignore them, and the 8+3 compatibility name is checksummed so the operating system can verify whether long name entries are valid or not.

Where Microsoft unconditionally permitted long file name extensions on all FAT file systems in Windows 95 and NT, without any user options to disable them, Linux instead separated out "msdos" and "vfat" drivers, allowing users to explicitly declare what kind of FAT compatibility is desired, the "msdos" retaining all the original limitations of MS-DOS names. This separation continues to exist in Linux versions to the present day.

UMSDOS was designed to only operate on the original pre-VFAT design of the file system (Linux's "msdos" file system), and can actually break in subtle ways when run in conjunction with Windows 95 or NT, systems which would tend to create long file name entries, sometimes even for things as simple as using mixed case in a name that would otherwise fit in 8+3 limits. This was one of the causes for UMSDOS's low adoption and decline around the turn of the century.[2]

1.3 Why UMSFUSE

UMSFUSE aims to provide a complete implementation of UMSDOS, providing both read and write capability to the file system. It achieves this goal through the use of Filesystem in Userspace (`https://github.com/libfuse/libfuse`) (FUSE), a combination of a Linux driver and library that allows for file system drivers to be rewritten and loaded as ordinary programs. One of the failings of the original UMSDOS implementation was its requirement to hook into and modify the Linux FAT driver to achieve its goals, and this proved itself to be fraught with danger, instabilities, and a high maintenance cost, leading to the eventual removal of UMSDOS in Linux 2.6.11.

With FUSE, instead, UMSDOS can be implemented in a safe manner, the on-disk FAT file system solely managed by the kernel and the extra Unix metadata UMSDOS provides can be managed such that it cannot corrupt the underlying file system. UMSFUSE is completely compatible with the original format, trees from old versions of Linux that wrote out UMSDOS structures can be read with all data and metadata intact. Likewise, UMSFUSE can create new UMSDOS trees that the old versions of Linux are able to read.

1.4 Compatibility goals

UMSFUSE is completely compatible, both for reading and writing, with all bits of metadata stored in the EMD files. It includes an extensive unit testing suite that tests various file name lengths, unusual names, hard links occupied in both the same directory and multiple directories, symbolic links, file modes (including sockets and pipes), and device files.

[2] The release of Windows XP caused the Windows NT File System (NTFS) to become common on home computers, and is another major factor in UMSDOS's decline.

A hand-crafted sample tree is routinely referenced in the unit tests, that was crafted specifically to catch edge cases of the UMSDOS format. Additional tests have also been devised as methods to break UMSDOS in the original Linux implementation were discovered.

1.5 What UMSFUSE provides

The primary component of UMSFUSE is '`umsdosfs`,' the FUSE file system driver that enables complete reading and writing of UMSDOS trees with all bits of metadata intact. See Section 3.1 [umsdosfs], page 10.

'`umsfsck`' can check and repair for damage and inconsistencies in the EMDs. It can even repair damages left behind by bugs in old versions of Linux. See Section 3.2 [umsfsck], page 12.

Other read-only data examination and extraction programs are detailed in Chapter 3 [UMSFUSE programs], page 10. They may be useful for copying out individual files or on systems where FUSE is unavailable: '`umscat`,' '`umscp`,' '`umsls`,' and '`umsstat`.' They provide roughly equivalent functionality to the same Unix programs without the *ums*-prefix.

`umstar` can create tar files from UMSDOS trees, as well as extract tar files directly into UMSDOS trees.

2 How it works

2.1 Host tree and the EMD

Every directory under UMSDOS contains a '`--linux-.---`' file, the Extended MS-DOS Directory (EMD). This is the file that holds all pertinent information describing the metadata associated with any particular file.

2.2 Unix names and mangling

In order to accommodate the vastly expanded capabilities of Unix file names and the restrictive requirements of MS-DOS names, UMSDOS uses a few rules to store each Unix file in a unique MS-DOS entry.

If a name is all-lowercase, fits within the 8+3 naming scheme, and the base name does not match a DOS device driver, it is stored on the hosting file system as-is.

If any of these rules are violated, UMSDOS will "mangle" the host file name. It will normalize the name to all-lowercase, special characters and spaces are converted to "`#`" characters, and a three-letter extension is chosen based on the address of the EMD slot the file is described in, using rules detailed in Section 4.3 [Name fields and mangling], page 24. Base names that match a DOS device driver (such as '`con`') are prefixed with a hyphen so they become a legal MS-DOS name.

2.3 Modes, owners and types

EMD files contain 16-bit entries to denote file modes, UIDs, and GIDs. They also contain 8-bit device minor and major values for device files.

File modes both affect the kind of file represented (directory, regular file, socket, named pipe, character and block devices, and symbolic links), as well as permission modes including the sticky bit, setuid, setgid flags, and the usual user, group, and other permissions.

2.4 Hard links

UMSDOS implements hard links in a peculiar way, one that largely breaks the expectation that MS-DOS names somewhat resemble those of the Unix names attached to them.

When a hard link is made on UMSDOS, the contents of the MS-DOS names for all names of the link will be replaced with a reference to a hidden link file. This hidden file is not visible from the view of a UMSDOS mount, but it is still described in the EMD and contains the canonical metadata retrieved through any of the hard link's names. The reference is relative to the UMSDOS root, and if subtrees of the UMSDOS tree are extracted and used, hard links may either reference the hidden link at the wrong path or the hidden link is missing altogether. Hidden link files are named '`..LINK`*n*', where *n* is a decimal offset of the EMD file's entry for it.

In the original UMSDOS implementation, the hidden link file is created within the directory of the original file. Users can trivially break the link and even crash Linux by renaming, removing, or moving a directory holding these links that names outside the tree still reference. Also in the original UMSDOS implementation, once a hard link is created, a hidden link persists even if the link count drops to 1.

UMSFUSE corrects both of these oversights by moving hidden link files up to a common ancestor, and "collapsing" a hidden link file back into a plain file once the link count drops to 1. The '`umsdosfs`' driver and '`umsfsck`' repair program can both modify file systems accordingly.

2.4.1 Old and new formats

There are two versions of the on-disk hard link format, an "old" format dating from UMSDOS's earliest days, up to and including Linux 2.2.14, and Linux 2.3.0 through 2.4.9.[1] The older format always prefixed hard link pointers with a "/," whose root depended on whether UMSDOS was mounted as Linux's root file system or a non-root file system. Root file systems would always start from Linux's VFS view, which started from the FAT's `/linux` directory. Non-root file systems would begin their reference strictly at the root directory of the FAT they lived on. This caused a discrepancy when file systems would be mounted in the two different ways and cause hard links to break between the modes.

The "new" hard link format, introduced with the UMSDOS driver's internal version 0.85g in Linux 2.2.15 and 2.4.10, omits a leading "/" from the pointers, and all references begin at the root of the FAT file system, whether mounted as a root directory or a non-root directory. For Linux installs using UMSDOS as root, all hard link pointers have `linux` prefixed before their complete paths, as this is where the Linux root lives on the FAT file system.

UMSFUSE defaults to using the "new" format, and supports the "old" format for reads and writes as well. Care should be taken to not mix both formats in a single file system, and if interoperability with an old Linux system is desired, take note of the kernel version being used. See Section 3.1 [umsdosfs], page 10, for mount options, and Section 3.8 [Common options], page 21, for options for all the other utility programs.

2.5 Symlinks

When a symbolic link is created on UMSDOS, the contents of the host file is the link target. Just as on native Unix file systems (eg, ext4), the link target need not exist, and in fact be any arbitrary string.

2.6 Timestamps

In the EMD, there is allocation for modification time (mtime), access time (atime), and change time (ctime). Each of these fields is 32-bit long, signed. This gives UMSDOS a date range from 1901-12-13 through 2038-01-19.

Given the upper ceiling of this date range draws close, at the time of writing, most modern systems have extended timestamps to a 64-bit range, extending the range from 136 years to approximately 584 *billion* years. UMSDOS, however, was written in the early 1990s where the ceiling was over 40 years in the future and considered a non-issue. See `https://en.wikipedia.org/wiki/Year_2038_problem` for details on this problem.

[1] Linux used to use odd numbers (like 2.3) to signify development branches. Linux forked from 2.2.8 to begin development on what would become 2.4.0. The 2.2.x branch continued to be maintained independently, with UMSDOS changes applied to both 2.2.15 and 2.4.10 at the same time.

To mitigate year 2038 problems, especially if you need to interact with UMSDOS after 2038-01-19, '`umsdosfs`' offers a '`-o umsclock=OFFSET`' option to shift the clock used for new timestamps, for all the timestamp types. See Section 3.1 [umsdosfs], page 10.

EMD records contain 12 unused bytes available for future expansion and an experiment in UMSFUSE was conducted to use these to extend all three timestamp fields to a full 64-bit representation. While it was successfully implemented in UMSFUSE, it posed compatibility issues with the original Linux driver and was abandoned. See the extended-timestamps branch (`https://fossil.chungy.name/umsfuse/timeline?r=extended-timestamps`) for details on the experiment.

2.7 Overmounting

Linux permits file system drivers (both kernel and FUSE) to "overmount" directories, where the view of the tree at that point is replaced by the driver mounting it, and it does not need to be empty first.

'`umsdosfs`' has been written and tested to safely overmount a UMSDOS tree, where the original host files become invisible to new processes and file access calls. It performs this by opening and not releasing the mountpoint prior to performing the mount, thereby giving a view of the underlying host file system for the driver even as new processes can only access the UMSDOS view.

2.8 Changes made directly to the host files

Changes made to a host file system do not update the UMSDOS metadata. Files added, removed, or renamed this way can therefore leave the host directory and the EMD out of alignment.

The historical UMSDOS utilities provided a '`umssync`' program to reconcile these differences, with Slackware preconfigured to run it on each boot. In UMSFUSE, this role instead belongs to '`umsfsck`,' which also checks the consistency of existing metadata and offers repairs.

Changing the host tree while '`umsdosfs`' has it mounted is not supported: the driver keeps directories in memory, and what it notices of such changes is described in Section 2.9 [Directory cache], page 6. Unmount first, change the host files, then run '`umsfsck`'.

2.9 Directory cache

'`umsdosfs`' keeps the directories it has read in memory, so that looking up, listing and `stat`ing names that were seen before costs no host access beyond the file itself. The first access to a directory reads it in full (the host listing and the whole EMD); afterwards a lookup is a hash table search, whatever the directory's size, and a listing a walk over the cached records.

Each cached directory holds:

- the host listing: every host name, indexed exactly and with ASCII case folded, so that a host name is found by the same rules as without the cache (an exact match first, else a single match ignoring case);

- the decoded EMD: one compact record per entry, indexed by its position in the EMD (which also gives the order of a listing and the free slots), by the Unix name, and by the host name with case folded.

Directories are identified by the host's device and inode number. Each cached directory is held open (an `O_PATH` descriptor), so its inode number cannot pass to another directory meanwhile (FAT gives inode numbers out afresh). The driver raises its open file limit to the hard limit at start, and caches at most that many directories less 256.

2.9.1 Memory

The cache is limited to '`-o umscache=SIZE`' bytes, 128 MiB by default; '`SIZE`' is a byte count with an optional '`K`', '`M`' or '`G`' suffix (powers of 1024). The count covers the records, the host names, the index tables and allocation overhead; on 64-bit hosts a directory of 1,000 entries with names of about 20 characters takes roughly 200 KiB. When an operation ends with the cache over its limit, the directories used least recently are dropped until it is within it again; the directories an operation is working on stay until it ends. A single directory larger than the whole limit is not cached; it is read from the host on each access, as it would be with the cache off, until it changes. '`-o umscache=0`' turns the cache off, and every access reads the host directory and its EMD again. The hard-link index (see Section 2.10 [Hard-link index], page 8) is counted in the same total.

2.9.2 Writes

Every change the driver makes to a host directory goes through a few central functions, which update the cached directory to match: names created, removed and renamed on the host, and the bytes written to an EMD, which are decoded into the cached records. Where the result cannot be followed exactly (an EMD rewritten in a way that does not map onto whole records, a host name that is not unique ignoring case, the EMD itself created or removed), the directory is read again from the host. The cache therefore does not depend on host timestamps for the driver's own changes, and the FAT time stamp granularity of 2 seconds does not matter for them.

2.9.3 Changes on the host while mounted

Before a cached directory is used, the driver compares the host's modification and change times of the directory and of its EMD (to the nanosecond), and the EMD's size and inode number, with those it recorded; if any differ, the directory is read again. After its own changes it records the new times. This catches most changes made to the host tree behind the driver's back, but not all of them:

- On FAT, directory times have a granularity of 2 seconds, and two changes within the same tick of the host's clock look like one anywhere; a change in the same tick as one the driver made itself goes unnoticed.
- A change to a host file's contents or attributes that does not change its directory is not a change of the directory: the cache holds names and EMD records, and file attributes are read from the EMD as always.
- The kernel caches names and attributes for about a second as well (FUSE's entry timeout), with or without the directory cache.

A directory read again shows the host as it is, with the same results as a fresh mount; changes it misses stay unseen until the directory changes again or is evicted. Either way the result is no different from the inconsistencies of Section 2.8 [Changes made directly to the host files], page 6: such changes are not supported while mounted.

For testing, the environment variable `UMSDOSFS_CACHE_NOCHECK=1` turns the comparison off, so that only the driver's own bookkeeping keeps the cache right, and `UMSDOSFS_CACHE_STATS=`*FILE* (an absolute path) appends the cache's counters to *FILE* at unmount.

'`umsfsck`' uses the same cache, with the same limit by default (see Section 3.2 [umsfsck], page 12); the host tree is not changed behind its back, as the file system must not be mounted meanwhile.

2.10 Hard-link index

Keeping hard links the way Section 2.4 [Hard links], page 4, describes needs to know every pointer to a hidden link file: a second link to a file that is already linked, removing the next-to-last name of a group (which collapses it), renaming a directory (pointers into it are rewritten) and renaming a hard-link name may each move a hidden file to a new common ancestor and rewrite its pointers. Without help, each of them reads every directory of the tree and every pointer in it, and then re-mirrors the host times of every record in the tree. With the directory cache on, '`umsdosfs`' keeps an index of the pointers instead.

2.10.1 Contents

One entry per pointer record: its host directory and record offset, its UMSDOS directory, its host name, the pointer text exactly as stored (either format, any link root or other leading components), and the hidden file it resolves to. The index is built by one scan of the whole tree, the first time a hard-link operation needs it, and is then kept up to date by every operation that adds, removes, renames or rewrites a pointer, moves or removes a hidden file, or renames a directory (the paths of everything below it follow).

It also counts every path component of every pointer text. A pointer is resolved by trying its text as a path, then with leading components stripped one at a time (see Section 3.1.2 [Hard link formats], page 11), so a pointer can come to name another hidden file when a name on one of those paths appears or goes. When an operation creates, removes or renames a name that some pointer text contains, those pointers are resolved again; one whose hidden file changed has its old and new groups checked as described below.

2.10.2 A hint only

The index is never trusted on its own:

- Before a pointer is acted on, its record and its host file are read again: the record must still be a pointer at the same offset with the same host name, the text must be the indexed text, and it must still resolve to the indexed hidden file. A pointer is never rewritten from what the index holds.
- A group (the pointers to one hidden file) must have exactly as many pointers as the hidden record's link count.
- When the directory cache's check finds a directory changed on the host behind the driver's back (see Section 2.9 [Directory cache], page 6), or a plain directory is given an EMD (which can change how names in it resolve), the index is dropped.

If any of these fails, the operation is done exactly as without an index (scanning the whole tree), and the index is built again from the tree afterwards. Changes to a pointer's contents made on the host while mounted are therefore found when that pointer, or its group, is next used; a change that leaves every check satisfied, such as a pointer added on the host to a directory the cache does not hold, goes unseen, as other changes on the host do (see Section 2.8 [Changes made directly to the host files], page 6).

2.10.3 Same results

For every operation the host tree comes out exactly as without the index: the same hidden files move, to the same directories, under the same `..LINKn` names, and the same pointers are rewritten with the same text. Where several hidden files move in one operation, they move in the order the scan would have found them, since that decides which EMD slots, and so which names, they get. The first operation that would have moved hidden files looks at every group, as before, which also settles any group the tree held out of place before the mount; later ones look only at the groups the operation touched.

Host times: after each such operation the whole tree's host times used to be re-mirrored from the records. With the index, the first one in a mount still does that (it also repairs times that drifted before the mount), as does any operation that fell back to the scan; the others re-mirror only the records they changed (rewritten pointers and moved hidden files), the directories they wrote being re-mirrored anyway.

2.10.4 Memory

The index is counted against '`-o umscache`', with the directories: about 200 bytes per pointer plus its text and paths. An index larger than half the limit is not kept, and the driver goes on scanning the tree for each operation, as with '`-o umscache=0`', which turns the index off along with the cache. The counters written with `UMSDOSFS_CACHE_STATS` include the index's size, how often it was built, and how often an operation fell back to the scan.

3 UMSFUSE programs

3.1 umsdosfs

`umsdosfs` ‘`SOURCE`’ [‘`MOUNTPOINT`’] [‘`OPTION`’...]

The ‘`umsdosfs`’ program is the FUSE driver that mounts a UMSDOS tree and exposes it as a fully-featured Unix tree, and all normal read and write operations work correctly. It is possible and safe for it to “overmount” a directory to transform its view from a host of EMD and mangled names into a standard Unix tree; see Section 2.7 [Overmounting], page 6. Mounting an empty directory is sufficient for creating a new UMSDOS tree.

Unmount a ‘`umsdosfs`’ file system with ‘`fusermount3 -u MOUNTPOINT`.’

‘`umsdosfs`’ expects that the host file tree has names in all-lowercase and is primarily intended to be used on top of Linux’s msdos driver, which ignores VFAT extensions and only provides 8+3 names. Case-insensitive file systems should work properly even if host file names are not all-lowercase, and ‘`Section 3.2 [umsfsck], page 12`’ can transform the tree into proper names on case-sensitive file systems, provided no case conflicts exist.

`SOURCE_DIR`
: A directory holding the UMSDOS tree.

`MOUNTPOINT`
: A directory to mount the UMSDOS tree at. The option may be omitted to create an overmount.

`-o ro`
`-r`
: Mount the file system as read-only.

`-o umsverbose`
`-v`
: Report EMD inconsistencies on stderr. Implies ‘`-f`’ because stderr is detached once the driver goes into the background.

`-o umsclock=OFFSET`
: Shift the clock used for new timestamps (new entries, writes, ctime, and touch without an explicit time); explicit times are stored as given. ‘`OFFSET`’ is a sum of signed terms, each number with an optional unit: ‘`s`’ (default), ‘`m`’, ‘`h`’, ‘`d`’, ‘`w`’, or ‘`y`’ (calendar years), e.g. ‘`-30y`’ or ‘`-1000d`’. It can also be an absolute base in UTC: an at sign followed by seconds since the epoch or by a date with an optional time, e.g. ‘`@631152000`’ or ‘`@1990-01-01T12:00:00`’. Fixed at mount time, refused if the shifted clock is outside the 32-bit range. Host files mirror the shifted record times.

`-o umslinkroot=PATH`
`--link-root=PATH`
: Hard link pointers carry `PATH` in front of the tree path. Leading and trailing slashes are ignored; `.` and `..` components are refused. Defaults to treating `SOURCE` as the root.

`-o umsoldlinks`
`--old-hard-links`
: Write hard link pointers in the old format (both formats are always read). Defaults to the new format. See Section 2.4.1 [Old and new formats], page 5.

`-o umscache=SIZE`
: Memory for the directory cache, in bytes with an optional '`K`', '`M`' or '`G`' suffix; defaults to '`128M`'. '`0`' turns the cache off. See Section 2.9 [Directory cache], page 6.

Standard FUSE options are also accepted:

`-f`
: Run the driver in the foreground.

`-d`
: Enable verbose debug output from libfuse3, implies '`-f`'.

`-s`
: Runs the driver in single-threaded mode, may be useful for debugging.

`-o allow_other`
: Allows other users to access the file system, by default only the user who mounts it can access it.

`-o default_permissions`
: Uses the kernel to enforce standard permission checks based on ownership and modes. By default all files are accessible and `chown`/`chgrp` work freely.

3.1.1 Deviations from the kernel

'`umsdosfs`' is compatible with, and provides the same logical view as the original UMSDOS driver from Linux versions prior to 2.6, but some of the low-level implementation details differ to provide a more stable experience.

Hard link data files are moved up the tree to the most common ancestor of all the hard link's names. The original UMSDOS would keep the data file in the same directory as the original name, and could easily break if the directory is renamed or moved. '`umsdosfs`' makes an active effort to keep all low-level references correct. '`umsdosfs`' will additionally collapse a hard linked file into a plain host file when the link count drops to 1. See Section 2.4 [Hard links], page 4.

'`umsdosfs`' also makes an active effort to reuse the first free slot of EMD files where room is available, on any operation that might write a new record (creation and renames). The original UMSDOS driver would instead always append to the EMD and never reclaim earlier space.

3.1.2 Hard link formats

Linux 2.2.15 and 2.4.10 introduced a breaking change in how hard links are written and interpreted. For UMSDOS root file systems, the old format treated the logical root directory as the basis for hard link pointers, and in the new format, the FAT root directory is always the basis for hard link pointers. Additionally, in the new format, a leading "/" is omitted from the reference path. See Section 2.4.1 [Old and new formats], page 5.

3.1.3 Limitations from UMSDOS

The following limits come from the original on-disk format and are maintained in '`umsdosfs`' by design:

- File names are limited to 220 bytes, rather than a more typical 255 bytes (or even 1023 bytes on some modern file systems). See Section 4.2 [Record format], page 23, for details on how this limit is derived.

- User and group IDs are limited to 16-bit values each, ranging from 0–65535. Most modern file systems support 32-bit UIDs/GIDs and container software often depends on the expanded capabilities, UMSDOS is incapable of hosting such containers.
- Device major and minor numbers are both 8-bit, ranging from 0–255.
- Timestamps are stored as 32-bit values, a signed number of seconds centered around 1970. Its range is 1901-12-13 20:45:52–2038-01-19 03:14:07 (both in UTC). See Section 2.6 [Timestamps], page 5.

  Setting a value outside these ranges (e.g. with '`touch`') will fail with '`EOVERFLOW`' and result in a no-op. Creating a file when the host clock is past 2038, and the '`-o umsclock`' option is not in use, will set the maximum possible positive value for the timestamps, which is 2038-01-19 03:14:07 UTC.
- Some file names that are valid on MS-DOS and FAT conflict with UMSDOS's own naming rules, such as backup files with the extension '`b~k`'. These are always renamed to a new mangled name.

3.2 umsfsck

`umsfsck` ['`OPTION`'...] ['`-u FILE`'] [`DIR`]

`umsfsck` '`--undo FILE`' ['`-n`'] ['`-v`'] [`DIR`]

'`umsfsck`' checks, repairs, and optimizes a UMSDOS file system. The file system must **not** be mounted in order to use '`umsfsck`.'

By default, it reports each found problem and asks '`(y/n/q)`' before fixing it. Press '`y`' to accept the fix, press '`n`' to decline the fix, and press '`q`' to decline the fix and cease checking the file system any further. The '`-y`' and '`-n`' options can automate the responses to these prompts.

'`umsfsck`' is able to check for the following issues:

EMD entry whose host file is missing
: An offer to remove the EMD entry is presented. This may happen when the host file is removed outside of a UMSDOS mount.

Host file not listed in the EMD
: Add an entry, with the owner, mode, and timestamps taken from the host file. The UMSDOS name is the file name in lowercase. If the name conflicts with a mangled name (e.g., '`foo.{__`'), it is renamed on the host to a new 8+3 name.

Host file with a valid MS-DOS name that UMSDOS mangles
: As above: an entry under that name is added to the EMD and the host file is renamed to a UMSDOS-compliant name.

Case-only host spelling difference for an existing EMD entry
: Both '`umsdosfs`' and '`umsfsck`' check to see if the host file system is case-insensitive, and do nothing as the software will function as expected on such a system. On a case-sensitive file system, renaming host file names to lowercase requires the '`-o`' option.

Host file name that breaks MS-DOS 8+3 rules
: A warning is issued and no fix is offered as such files cannot be referenced by UMSDOS.

Hard link pointer that cannot be resolved
: Looks for the intended target. If one is found, it offers "Point X at Y?," even when Y is the only candidate. If the target cannot be determined, nothing is changed, even with '`-y`'. Interactive mode lists the candidates by number to pick from, or '`r`' to remove the link. If no hidden file could be the target, it offers to remove the dangling link.

  This check might result in false positives if the proper UMSDOS root is not being checked (e.g., checking a UMSDOS subtree).

Hidden link file with the wrong link count
: Correct the '`nlink`' field in the EMD. Left uncorrected while an unresolved link might refer to this file.

Hidden link that no pointer references
: If one plain record has its metadata and host file holding a link path to it, it is an interrupted collapse (see Section 2.4 [Hard links], page 4): offer to finish it by moving the data onto that name and clear the hidden record. Otherwise, turn it into a visible regular file (no data is deleted.)

Record running past the end of the EMD
: Clear the record.

EMD size is not a multiple of 64
: Issue a warning, leave uncorrected.

Duplicate names, or the entry type does not match host (directory versus regular file), and illegal names
: Reported and left uncorrected.

EMD is in a non-i386 format
: Report as not repairable, that directory and all subdirectories are skipped, no repair is ever offered.

An undo file can be generated with '`-u`', and applied with '`--undo`', to restore a UMSDOS tree to its original state before any and all changes by '`umsfsck`' were made.

`-n`
: Dry run: report everything that '`-y`' would do, working on a scratch copy of the tree in '`$TMPDIR`'. The copy has every host name, mode and time, but file data only where '`umsfsck`' reads it: control files, hard-link pointers, symlinks, and a file that may be the survivor of an interrupted collapse. Every other host file is copied empty, so the copy needs little room however large the files are. Never writes to the UMSDOS tree being checked. An exit code of 4 is returned if any corrections could be made.

`-y`
: Answer "yes" to every fix whose result is unambiguous. '`-n`' and '`-y`' cannot be combined.

`-o`
: Optimize the file system. Hard links with only one remaining link are collapsed into a plain file holding the data. EMD files are shrunk to their smallest possible size, host files are renamed to conform to the new EMD locations. '`..LINK`*n*' files that move are renamed to their new offset, and hard links referencing them are rewritten. Timestamps in the EMDs are mirrored to the host file system as

best possible. Each directory's planned moves, renames, and time changes are listed, then asked for like any fix ('`-n`' and '`-y`' can be used to automate the optimization report and fixes).

Optimization is skipped if any problems remain unresolved, an operational error occurred, or the run was cancelled. It does not change the exit status.

`-v` Verbose operation.

`-u FILE` Before each change to the tree, record in `FILE` how to reverse it: the states of the host objects touched and the bytes the change destroys. Off by default. `FILE` must not exist and must not be inside the tree. Nothing is logged with '`-n`'.

`--undo FILE`
Reverse the changes recorded by '`-u FILE`' to the original state of the UMSDOS tree. Refused if any object the run touched was changed in the interim. Safe to run again after an interruption. Combined with '`-n`', only check and list the undo steps.

`-s` Strict mode: hard link references that only resolve after stripping leading path components count as errors, and offered to rewrite relative to `DIR`. Use it when `DIR` is the root of the original UMSDOS file system.

On `SIGINT` (*`C-c`*), `SIGTERM` or `SIGHUP`, '`umsfsck`' stops at the next point where that is safe, finishes an undo file ('`-u`'), and ends with the signal, so the shell reports it (exit status 130 for *`C-c`*). With '`-n`', it first removes its scratch copy. Otherwise the repair under way is completed and none is started after it; during '`-o`', the directory being optimized is completed, the remaining directories are left as they are, and hard links to hidden files renamed so far are rewritten first. Each repair leaves the tree consistent, so running '`umsfsck`' again finishes the job. A signal that was ignored when '`umsfsck`' started (as by `nohup`) stays ignored.

'`umsfsck`' reads directories through the same cache as '`umsdosfs`' (see Section 2.9 [Directory cache], page 6): each directory is read from the host once, and names and records are then looked up in memory, so a check takes time in proportion to the size of the tree, whatever the number of entries per directory or of hard links. The cache holds 128 MiB by default; directories beyond that are dropped and read again when needed, which is slower but gives the same results. Its own changes keep the cache up to date, as in the driver. With '`-n`' the cache is used only on the scratch copy. The copy reads each host directory's listing once and answers its own lookups of host names from it, whatever their case, so it too takes time in proportion to the size of the tree; it remains the largest part of a dry run's time, as it creates every host name in the copy (with the data of the files listed above). Beyond the cache, memory grows with the number of hard links and directories and the size of the largest directory, not with the size of the tree.

The environment variable `UMSFSCK_CACHE=`*`SIZE`* (a byte count with an optional '`K`', '`M`' or '`G`' suffix) sets the cache's limit, and '`0`' turns it off, so that every lookup reads the host directory again; this is much slower on large directories, and the results are the same. For testing, `UMSFSCK_CACHE_NOCHECK=1` and `UMSFSCK_CACHE_STATS=`*`FILE`* work like their '`umsdosfs`' counterparts; the counters are appended at the end of the run.

3.3 umscat

`umscat` [‘`OPTION`’...] [‘`--`’] `FILE`...

‘`umscat`’ concatenates files, using names from UMSDOS metadata, and copies their contents to stdout. By default, the current directory is used as the UMSDOS root and paths are relative to it. Symbolic and hard links are followed; directories and other non-regular files are reported as errors.

This utility operates directly with UMSDOS metadata files, and not from a mounted tree, where the standard ‘`cat`’ command may be used instead. It will refuse to run on a ‘`umsdosfs`’ mount.

3.4 umscp

`umscp` [‘`OPTION`’...] [‘`--`’] `SOURCE... DEST`

‘`umscp`’ copies files, using names from UMSDOS, and writes them to the specified destination on the normal file system. By default, the current directory is used as the UMSDOS root and paths are relative to it. If the destination is an existing directory, each source is copied into it under its own name. Otherwise a single source file is copied to the destination name.

‘`umscp`’ operates much like ‘`cp -p`:’ mode, ownership, timestamps, symbolic links, hard links, FIFOs (named pipes), and devices are preserved as much as possible. Ownership and device nodes normally require root to alter, and a warning is printed if they cannot be preserved.

This utility operates directly with UMSDOS metadata files, and not from a mounted tree, where the standard ‘`cp`’ command may be used instead. It will refuse to run on a ‘`umsdosfs`’ mount.

`-R`
`-r` Copy directories recursively.

`-v` Verbose operation, printing each file as it is copied on stderr.

`-N` Do not preserve ownership.

`-L` Dereference specified UMSDOS symbolic links.

3.5 umsls

`umsls` [‘`OPTION`’...] [‘`--`’] `PATH`...

‘`umsls`’ lists information about the paths specified, using the UMSDOS metadata for names, modes, and other information. It may also display the associated mangled name for UMSDOS files, making it easy to interoperate with them. By default, the current directory is used as the UMSDOS root and paths are relative to it. Names are sorted with the locale’s collation and, like GNU ‘`ls`’, listed in columns on a terminal and one per line otherwise (see [umsls column layout], page 17).

This utility operates directly with UMSDOS metadata files, and not from a mounted tree, where the standard ‘`ls`’ command may be used instead. It will refuse to run on a ‘`umsdosfs`’ mount.

`-l` Display a long listing: mode, link count, owner, group, size (or device numbers), modification time and name, followed by '`->`' and the target for a symbolic link.

`-a` Also show hidden hard link base file entries (the `..LINK`*n* files), marked by "(hidden)." With '`-l`' and without '`-m`', a hard link is followed by '`=>`' and the text stored in its pointer file.

`-m` Display the mangled host name attached to the file. For a hard link, also where its data contents reside. See Section 2.2 [Unix names and mangling], page 4.

`-R` Recursive listing.

`-d` List directories as themselves, not their contents.

`-i` Display inode numbers.

`-n` Display numeric UIDs and GIDs; implies '`-l`'.

`-U` List files unsorted, in the EMD order.

`-x` List in columns, sorted across the rows instead of down the columns.

`-1` List one entry per line.

`--format=WORD`
Choose the layout: '`vertical`' (columns sorted down, the default on a terminal), '`across`' or '`horizontal`' ('`-x`'), '`commas`' (names separated by '`, `', filling each line), '`single-column`' ('`-1`', the default otherwise), '`long`' or '`verbose`' ('`-l`'). An unambiguous abbreviation is accepted.

`-w COLS`
`--width=COLS`
Lay out the columns for a line width of COLS. 0 means no limit: the columnar formats then list a whole directory on one line.

`-T COLS`
`--tabsize=COLS`
Assume tab stops every COLS columns (default 8). 0 pads the columns with spaces only.

`-N`
`--literal`
Print entry names without quoting.

`-b`
`--escape` Print C-style escapes for nongraphic characters.

`-Q`
`--quote-name`
Enclose entry names in double quotes.

`-q`
`--hide-control-chars`
Print '`?`' instead of nongraphic characters.

`--show-control-chars`
Print nongraphic characters as-is.

`--quoting-style=WORD`
: Use the quoting style WORD for entry names: literal, locale, clocale, shell, shell-always, shell-escape, shell-escape-always, c, c-maybe, escape (overrides '`QUOTING_STYLE`' environment variable).

Column layout

The layouts follow GNU '`ls`' 9.12: on the same names, in the same locale and with the same options, '`umsls`' prints the same bytes as '`ls`' does for an ordinary directory.

- The format is chosen by the last of '`-l`', '`-n`', '`-x`', '`-1`' and '`--format`', except that '`-1`' after '`-l`' leaves the long format. Without one of them the output is in columns sorted down on a terminal, and one per line otherwise. The long format is always one per line. '`-C`' is not the '`ls`' option: like in the other tools, it gives the tree root. For columns in a pipe, use '`--format=vertical`'.
- The line width is '`-w`', else the terminal's width (when the output is a terminal), else the `COLUMNS` environment variable, else 80. It is an inclusive maximum: a line may be exactly that wide. An invalid `COLUMNS` is ignored with a warning. The tab size is '`-T`', else the `TABSIZE` environment variable, else 8.
- The number of columns is the largest for which every line fits: each column is as wide as its widest entry and two spaces, the last column without the spaces. Columns are padded with tabs where a tab reaches the next column and with spaces otherwise, as '`ls`' does.
- Widths are display widths in the locale (`wcwidth`), counted on the names as they are shown, after quoting. As in '`ls`', with the shell, shell-escape and c-maybe quoting styles an unquoted name gets one space in front when a name in the same listing is quoted, in the columnar formats as in '`-l`'. A name printed with raw control characters or bytes invalid in the locale (the literal style when not on a terminal, or '`--show-control-chars`') counts a control character as 0 columns and an invalid byte as 1; GNU '`ls`' counts such a name as `SIZE_MAX` columns, which wraps around and misaligns its columns, and this is not copied.
- Each entry is laid out as one cell: everything shown for it apart from the newline, including the inode number of '`-i`', the '`(hidden)`' mark of '`-a`' and, with '`-m`', the bracketed host name and, for a hard link, the '`=>`' part. Inside the brackets the host name keeps its padding to 12 columns, so a cell reads the same as the line '`-1`' shows for the entry, and as every column starts at the left of its cells, the names after the brackets line up within each column. For example, at 80 columns:

```
[long#fil.{__]  Long file name.txt  [readme      ]  readme
[makefile.{_6]  Makefile            [with#spa.{_2]  with space
[upper.{_3   ]  UPPER               [x_tar_gz.{_5]  x.tar.gz
[a#b(c).txt  ]  a#b(c).txt
```

  A '`-m`' cell is thus at least 16 columns wide, and a hard link's some 50, so these listings get fewer columns; a cell wider than the line leaves a single column, the same as '`-1`'.
- Arguments that are not listed as directories (files, and directories with '`-d`') are listed first, together, in the chosen format, then each directory, preceded by a blank line and its UMSDOS path followed by '`:`' when there is more than one argument or with '`-R`'.

Both groups are sorted unless '`-U`' is given. Each directory, including each one reached by '`-R`', is laid out by itself.

3.6 umsstat

`umsstat` ['`OPTION`'...] ['`--`'] `PATH`...

'`umsstat`' displays metadata information about each specified path, according to its record in the EMD: the mangled host name and host path, the EMD file, the record's offset, length and flags, and the link count, mode, owner, device numbers, host file size and times stored in it. Times are shown in local time and in seconds. A symbolic link also shows its target.

For a hard link, the record shown is the link pointer; it is followed by the text stored in the pointer file, the hidden `..LINK` file it resolves to, and that file's record, indented. A hidden file can also be named directly, by its UMSDOS name or its host name.

The tree root, and entries of a directory without an EMD, have no record. Their host attributes are displayed instead. By default, the current directory is used as the UMSDOS root and paths are relative to it. Names are quoted as by '`umsls`.'

This utility operates directly with UMSDOS metadata files, and not from a mounted tree. It will refuse to run on a '`umsdosfs`' mount.

`-L` Dereference symbolic links.

`-x` Display the raw bytes of each record, with their offsets in the EMD.

`-c FORMAT`
`--format=FORMAT`
: Print '`FORMAT`', followed by a newline, instead of the default output. Directives describe each entry's own record (for a hard link, the pointer).

`-H` With '`-c`', describe a hard link's hidden file instead of the pointer.

`--quoting-style=WORD`
: Quote names in the given style, as in '`umsls`.'

Format directives may have '`printf(3)`' flags and a width, such as '`%-8N`'.

`%n` Path as given.

`%N` UMSDOS path, quoted.

`%M` Mangled host name.

`%H` Host path from the root.

`%E` Host path of the EMD.

`%p` Record offset in the EMD.

`%P` Record length, in bytes.

`%e` Flags: 1 is hidden, 2 is a hard link.

`%k` Link pointer text.

`%K` Hidden file, UMSDOS path.

`%I` Hidden file, host path.

`%h` Link count.

`%a`
`%A` Access rights, octal and human-readable.

`%f`
`%F` Raw mode in hexadecimal, and file type.

`%u`
`%U` User ID and name.

`%g`
`%G` Group ID and name.

`%t`
`%T` Major and minor device numbers, in hexadecimal.

`%s` Size of the host file.

`%x`
`%y`
`%z` Access, modification, and change timestamps.

`%X`
`%Y`
`%Z` Access, modification, and change timestamps, in seconds since 1970-01-01 00:00:00 UTC.

`%%` A literal %.

3.7 umstar

`umstar` [‘`OPTION`’...] [‘`--`’] [‘`PATH`’...]

‘`umstar`’ writes or extracts a POSIX tape archive (ustar, with pax extended headers where ustar falls short) of a UMSDOS host tree, without mounting it.

It supports just two major modes of operation:

Create (‘`-c`’, default)
: The contents of the whole tree are archived (the root itself has no record and is not an entry unless ‘`-r`’ is given). Each ‘`PATH`’ is a UMSDOS path relative to the root, and is archived under that name, without a leading `/`; directories are archived recursively.

  Members are always in the order of GNU `tar`’s ‘`--sort=name`’: the names in each directory are sorted by byte value, as in the C locale, each directory followed by its contents. ‘`PATH`’ arguments keep the order given.

  Entries carry UMSDOS names and the mode, uid, gid, and mtime of their records. Symbolic links, character and block devices, FIFOs and directories are archived as such. Sockets are skipped with a warning, being unable to be recorded in an archive file. Names of a hard link group share one data file: the first name archived holds the data, later names are hard link entries. The

hidden `..LINKn` files and EMD files are never archived. Directories without an EMD are archived with their host names and attributes.

Every entry has a pax header with the atime and ctime of its record. It also holds what ustar cannot: a path cannot be split into 155+100 bytes, a link target over 100 bytes (with `hdrcharset=BINARY` if either is not UTF-8), or an mtime before 1970.

Extract (‘`-x`’)
: Members are written into an offline UMSDOS tree at ‘`-C ROOT`’, defaulting to the current directory: host files, mangled 8+3 names, and EMD records using the same rules as ‘`umsdosfs`.’ Hard links, devices, FIFOs, and symbolic links are restored. Pax atime and ctime are stored in the EMD when present; if either are missing, the member's mtime is used for the missing fields.

  A member that cannot be represented in UMSDOS is skipped with a warning (exit status 1): a name longer than 220 bytes, the exact name `--linux-.---` (case variants such as `--LINUX-.---` are extracted), a name containing DEL (0x7f), `.` or `..`, or a socket. Other members are still extracted. `PATH` arguments limit extraction to those UMSDOS paths and their descendants.

‘`umstar`’ does not directly support compression wrappers, a separate utility such as ‘`gzip`’ or ‘`zstd`’ must be used to handle it externally, for example:

Creating a compressed archive:
```
umstar | zstd > ~/umsdos-backup.tar.zst
```

Extracting a compressed archive:
```
zstdcat ~/umsdos-backup.tar.zst | umstar
```

This utility operates directly with UMSDOS metadata files, and not from a mounted tree. It will refuse to run on a ‘`umsdosfs`’ mount.

`-x`
`--extract`
: Extract the archive into the UMSDOS tree at ‘`-C ROOT`’.

`-f FILE`
`--file=FILE`
: Archive file to create or extract. Defaults to stdout for create, or stdin for extraction.

`-r DIR`
: Create mode only: put every member under `DIR` and archive `DIR` first as a directory with the tree root's mode, owner, and mtime, as ‘`umsdosfs`’ shows the root. `DIR` must be relative and without `..` or `.` components. Repeated and trailing slashes are dropped.

`-u`
: Create mode: fill in owner and group names from this host's user and group databases. For extraction, if a member carries a user or group name, resolve it on this host (numeric IDs from the header are used when a name is absent or unknown).

  By default, only numeric IDs are stored or applied.

`-U` Like '`-u`', but fill in owner and group names from the UMSDOS tree's own `/etc/passwd` and `/etc/group` databases.

`-L` Create mode only: dereference UMSDOS symbolic links given as `PATH`.

`-v` List member names on `stderr`.

3.7.1 Interoperability

GNU '`tar`' 1.35 warns that it ignores the `hdrcharset` keyword; names are still extracted byte for byte. libarchive's '`bsdtar`' strips a leading drive letter such as `C:` from names unless given the '`-P`' option.

3.8 Common options

These options are common to the entire suite of UMSFUSE programs:

`-h`
`--help` Displays a usage summary and brief help descriptions.

`-V`
`--version`
Displays the version of the UMSFUSE utility.

`-C ROOT` Sets the UMSDOS root directory, useful especially to resolve hard links correctly.

`--link-root PATH`
Hard link pointers carry `PATH` before the tree path. This is useful especially for UMSDOS-root systems, where you might need to specify `linux` for the `PATH` to have hard links work properly.

`--old-hard-links`
Write hard link pointers in the "old" format (pre-2.2.15, pre-2.4.10).

`--` Stops processing program options and treats all further arguments as file names.

All UMSFUSE programs expect the host file system to behave like the "msdos" file system, and it is best used through an "msdos" mount. Reads and writes are done exclusively through lowercase file names. '`umsfsck`' has the capability to repair a tree on a case-sensitive file system by changing all the host names to lowercase. Names on case-insensitive file systems are left alone.

4 UMSDOS specification

This chapter describes the UMSDOS format as it is stored on a FAT file system: the layout of the EMD files, how a record names its host file, what each field means, and how hard links, symbolic links and device files are represented. It is meant to be detailed enough to write a reader, or to make sense of the bytes of an EMD with a hex dump.

The format is the one the UMSDOS driver of Linux 2.x on i386 reads and writes. The descriptions of that driver here come from the Linux sources (mainly 2.2.26 and 2.4.31, with earlier releases where noted), from the UMSFUSE sources, which encode the layout UMSFUSE reads and writes, and from the sample trees in the UMSFUSE test suite. Where UMSFUSE deliberately behaves differently from Linux 2.x, the text says so ("UMSFUSE deviation"). Points that have not been checked against a running Linux 2.x system are marked as unverified.

All offsets and sizes are in bytes, offsets counted from 0. All multi-byte values are little-endian. "Linux 2.x" means the UMSDOS driver of the Linux 2.2 and 2.4 kernels; FAT names are written in lower case, as Linux's '`msdos`' driver shows them.

4.1 Directory layout

A UMSDOS tree is an ordinary FAT directory tree. A directory that UMSDOS manages, a *promoted* directory, contains a regular file named `--linux-.---`: its EMD. The name is a legal 8+3 name, so DOS sees the EMD as an ordinary file (stored as `--LINUX-.---` in the FAT directory entry). Lookups of the EMD, like all host lookups, ignore case; UMSFUSE creates it in lower case and accepts any case. This applies to the host name only: as an entry name, only the exact lower-case `--linux-.---` is reserved (see Section 4.3 [Name fields and mangling], page 24).

When a directory has an EMD, the EMD *is* the directory. Only names that have a record in the EMD are visible, and an EMD that holds no records makes the directory look empty, whatever host files it contains. Each record describes one entry and names one host object (a file or directory) in the same host directory; the host name is derived from the record's name and its position in the EMD (see Section 4.3 [Name fields and mangling], page 24). Host files without a record are invisible. The entries `.` and `..` are never recorded: FAT provides them.

A directory without an EMD is a plain DOS directory. Its host files are shown under their host names with the attributes the FAT driver gives them, and Linux 2.x allowed no Unix features in it (no long names, links, device files or FIFOs). Linux 2.x created an EMD in every directory made with `mkdir` inside a promoted directory, and the `umssync` utility promoted existing directories. UMSFUSE does the same on `mkdir`. *UMSFUSE deviation:* the first operation that adds a name to a plain directory promotes it first, giving it an EMD that lists the host files already there.

The tree root has no record of its own: a directory's record lives in its parent's EMD, which is outside the tree. The root's metadata comes from the host directory.

When Linux 2.x mounted UMSDOS as its root file system and the FAT held a `\LINUX` directory with Linux in it (2.4.31 looks for `\LINUX\SBIN\INIT`), it used a *pseudo-root*: `\LINUX` became `/`, and the FAT root appeared as `/DOS`. This matters to the on-disk format

only through hard-link pointers (see Section 4.6 [Hard-link records], page 28). UMSFUSE has no pseudo-root; its tree root is the directory it is given.

4.2 Record format

An EMD is a sequence of 64-byte *slots*. Its size should be a multiple of 64; UMSFUSE ignores a trailing partial slot, and `umsfsck` reports it.

A slot whose first byte is 0 is free. Any other first byte is the name length of a record starting in that slot. A record is a 36-byte header followed by the name, and occupies as many consecutive slots as it needs:

```
slots = (36 + name_len + 63) / 64        (integer division)
```

so names of 1–28 bytes take one slot, 29–92 two, 93–156 three and 157–220 four. A record is at most 256 bytes long. To walk an EMD, start at offset 0; at a free slot advance by 64 bytes, at a record advance by its number of slots times 64.

The header, in the i386 layout:

Offset	Size	Field	Contents
0	1	`name_len`	Length of the name, 1–220; 0 marks a free slot.
1	1	`flags`	`0x01` hidden, `0x02` hard-link pointer; see Section 4.4 [Metadata fields], page 26.
2	2	`nlink`	Link count, unsigned.
4	2	`uid`	Owner, unsigned.
6	2	`gid`	Group, unsigned.
8	4	`atime`	Access time, signed seconds since 1970-01-01 00:00:00 UTC.
12	4	`mtime`	Modification time, signed, as `atime`.
16	4	`ctime`	Change time, signed, as `atime`.
20	1	`rdev` minor	Device minor number.
21	1	`rdev` major	Device major number.
22	2	`mode`	File type and permissions, Linux values.
24	12	`spare`	Unused; always written as zeros.
36	*name_len*	`name`	The name, not terminated.

This is `struct umsdos_dirent` of Linux 2.x with the i386 sizes of its types (16-bit `uid_t`, `gid_t`, `dev_t` and `umode_t`, 32-bit `time_t`); Linux converts every field to little-endian when it writes it. Bytes 20 and 21 together are the little-endian 16-bit device number `major * 256 + minor`. The bytes after the name, up to the end of the record's last slot, are written as zeros. When Linux 2.x frees a record, it overwrites all of its slots with zeros, and so does UMSFUSE. The header comment of `struct umsdos_dirent` calls `ctime` the creation time, but the driver maintains it as the inode change time.

As an example, these are the first two slots of the root EMD of the sample tree in the test suite: a directory `control-chars` at offset 0 and a hard-link pointer `link1.txt` at offset 64.

```
0000  0d 00 01 00 00 00 00 00  ff ed b7 2e ff ed b7 2e
0016  ff ed b7 2e 00 00 ed 41  00 00 00 00 00 00 00 00
0032  00 00 00 00 63 6f 6e 74  72 6f 6c 2d 63 68 61 72
0048  73 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
0064  09 02 01 00 00 00 00 00  5c ee b7 2e 5c ee b7 2e
0080  5c ee b7 2e 00 00 ff 81  00 00 00 00 00 00 00 00
0096  00 00 00 00 6c 69 6e 6b  31 2e 74 78 74 00 00 00
0112  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

The first record has `name_len` 13, no flags, `nlink` 1, owner and group 0, all three times `0x2eb7edff` (1994-11-02 19:52:31 UTC), no device number and mode `0x41ed` (octal `040755`, a directory with permissions `0755`). The second has `name_len` 9, flag `0x02` and mode `0x81ff` (octal `0100777`).

Slot allocation

A new record goes into the first run of consecutive free slots that is large enough, or is appended at the end of the EMD if there is none. Linux 2.x does this (`umsdos_find` in 2.2.6, 2.2.26 and 2.4.31), and so does UMSFUSE. When Linux 2.x renames an entry, it writes the record for the new name before it frees the old one, so the new record never takes the old one's slots. *UMSFUSE deviation:* a rename within one directory rewrites the record in place when the new name fits in the slots the record already has, and frees the slots a shorter name no longer needs. Because a mangled host name depends on the record's offset (see Section 4.3 [Name fields and mangling], page 24), moving a record to another offset means renaming its host file.

4.3 Name fields and mangling

The name field holds the entry's Linux name: 1–220 bytes, compared byte for byte (case-sensitive), in no particular character encoding. It cannot contain a NUL byte, '`/`' or DEL (`0x7f`), and it cannot be `.`, `..` or exactly `--linux-.---`, the EMD's own name. Linux 2.x silently truncated a longer name to 220 bytes; *UMSFUSE deviation:* UMSFUSE refuses it with `ENAMETOOLONG`.

Linux 2.x refuses to create a name containing DEL, and the name `--linux-.---`, with `EIO` (`umsdos_parse` contains code to mangle `--linux-.---` like any other name, but creating it fails all the same). UMSFUSE refuses both with `EINVAL`, as it does other names it cannot store. Only the all-lower-case spelling is reserved: `--LINUX-.---`, `--Linux-.---` and every other case variant are ordinary names in both. They contain upper-case letters, so they are mangled like any such name (`--LINUX-.---` at slot 0 gets the host name `--linux-.{__`), and their host files never coincide with the EMD, even on FAT, where the EMD's directory entry is `--LINUX-.---`.

Deriving the host name

Every record's host object is the file or directory, in the same host directory, whose name is computed from the record's name and offset as follows. This is `umsdos_parse` and `umsdos_manglename` of Linux 2.x, which UMSFUSE follows exactly.

1. A translation table T maps each byte to a byte acceptable in a DOS name: `0x00`–`0x20` (control characters and space) to '`#`'; '`A`'–'`Z`' to lower case; '`.`' to '`_`'; the characters `" * + , / : ; < = > ? [ \ ] | ~` and the bytes `0x80`–`0xff` to '`#`'; every other byte from `0x21` to `0x7e` to itself. A byte b is *acceptable* when $T[b]$ equals b. T[`0x7f`] is never used, since a name containing DEL cannot be created (Linux 2.x's table leaves it 0; UMSFUSE's maps it to '`#`').

2. The name is used unchanged as the host name, without mangling, when all of these hold: it is at most 12 bytes long; it does not start with '.'; it holds at most one '.'; with a '.', the part before it is at most 8 bytes, the part after it is 1–3 bytes, and a 3-byte extension does not start with one of `{ } ( ) ! ` ^ & @`; without a '.', it is at most 8 bytes; every other byte is acceptable; and it is not exactly `--linux-.---` (a name that cannot be created anyway). In practice: an all-lower-case, legal 8+3 name whose extension cannot be mistaken for a mangled one.
3. Otherwise the name is *rejected*. The host base name is the first 8 bytes of the name (all of it, if shorter), each translated by T; it has no '.', since T turns dots into underscores.
4. The base name is then checked against the DOS device names `prn`, `con`, `aux`, `nul`, `lpt1`–`lpt4`, `com1`–`com4`, `clock$`, `emmxxxx0`, `xmsxxxx0` and `setverxx`. The base name is the part before the dot for an unchanged name, and the whole 1–8 byte host name for a rejected one. If it equals one of them, a '`-`' is put in front (and the result cut to 8 bytes, so `emmxxxx0` becomes `-emmxxxx`); for an unchanged name, its extension is dropped. The name then counts as rejected.
5. A rejected name gets an extension computed from the number n of the record's first slot (its offset divided by 64):

   ```
   .  L3[(n >> 10) & 31]  L12[(n >> 5) & 31]  L12[n & 31]
   ```

   where *L12* is the 32 characters `_123456789abcdefghijklmnopqrstuv` ('`_`' standing for zero) and *L3* is the 9 characters `{}()!`^&@` followed by those of *L12*. Slot 0 gives `.{__`, slot 1 `.{_1`, slot 32 `.{1_` and slot 1024 `.}__`.

Some results, for records at the offsets given:

Name	**Offset**	**Host name**
`readme.txt`	any	`readme.txt`
`README.TXT`	0	`readme_t.{__`
`Hello.c`	64	`hello_c.{_1`
`.profile`	128	`_profile.{_2`
`a.b.c`	0	`a_b_c.{__`
`con.txt`	192	`-con.{_3`
`config.b~k`	0	`config_b.{__`
`--Linux-.---`	64	`--linux-.{_1`
`foo.{__`	0	`foo_{__.{__`
`C:\DOS`	256	`c##dos.{_4`
`..LINK128`	128	`__link12.{_2`

The last two are records of the sample tree. As the `config.b~k` row shows, a legal DOS name can still be rejected: any name with '`~`' or a byte `0x80`–`0xff`, or with a 3-character extension starting with one of the characters above, gets a mangled host name, so a host file with such a name can only be listed under a different host name.

Within one EMD, two records never get the same host name as long as the slot numbers stay below 9216 (9 * 32 * 32, an EMD smaller than 589,824 bytes): every mangled name has a first extension character from *L3*'s first 9 characters, which no unchanged name can have, and mangled names differ in their slot. From slot 9216 on the first character comes from *L12*, so a mangled name can equal an unchanged one (slot 9216 gives `.___`), and from slot 32768 the extensions repeat. Linux 2.x prints a warning beyond 9216 slots. UMSFUSE refuses to create an entry whose host name is already taken (`EEXIST`).

Looking up a name

To look up a name in a promoted directory: walk the EMD for a record whose name has the same length and bytes; compute its host name from the name and the record's offset; and look up that host name in the host directory, ignoring case.

Linux 2.x matches hidden records in a lookup as well; it only leaves them out of directory listings. If the host object is missing, Linux 2.x silently deletes the record and reports that the name does not exist. *UMSFUSE deviation:* the mounted view does not find hidden records by name, and a record whose host object is missing is reported as missing and left for `umsfsck` to repair.

4.4 Metadata fields

`flags` Bit `0x01` (`UMSDOS_HIDDEN`): the entry is never listed. It marks the data file of a hard-link group. Bit `0x02` (`UMSDOS_HLINK`): the entry is a hard-link pointer (see Section 4.6 [Hard-link records], page 28). Ordinary entries have no flags. No other bits are defined, and a record never has both.

`nlink` For a hidden hard-link data file, the number of pointers that refer to it. For other non-directories, 1, which pointers also carry. For a directory the field is not meaningful: Linux 2.x takes a directory's link count from the FAT driver, writes 1 when it creates a directory (2.4.31) and later whatever count it holds in memory when the record is updated. UMSFUSE writes 2 for a new directory and shows the host directory's link count. It shows a non-directory record with `nlink` 0 as having one link.

`uid`
`gid` The owner and group, 0–65535. UMSFUSE refuses larger values with `EOVERFLOW`.

`mode` The Linux `st_mode`: the file type in the bits `0170000` (`0140000` socket, `0120000` symbolic link, `0100000` regular file, `060000` block device, `040000` directory, `020000` character device, `010000` FIFO) and the setuid (`04000`), setgid (`02000`), sticky (`01000`) and permission bits. Any other file type is invalid. The type tells how to read the host object (see Section 4.7 [Special files], page 29).

`rdev` For character and block devices, the minor number in byte 20 and the major number in byte 21, each 0–255. UMSFUSE writes zeros for other types and refuses larger numbers with `EOVERFLOW`.

`spare` Twelve bytes, written as zeros by Linux 2.x whenever it writes a whole record, and always by UMSFUSE. Readers should ignore them. (Linux 2.x updates the header of an existing record in place, which leaves them as they were.) The detection of non-i386 layouts relies on bytes 28 and 29 being zero in i386 records (see Section 4.8 [Foreign layouts], page 30). An UMSFUSE experiment that stored extended timestamps here was abandoned (see Section 2.6 [Timestamps], page 5); no current program uses these bytes.

The atime, mtime and ctime fields are described in Section 4.5 [Timestamps in detail], page 27.

Host permissions

The host file's own attributes are not part of the record, but Linux 2.x kept one of them in step with it: when the FAT driver wrote an inode back, it set the FAT read-only attribute (`ATTR_RO`) exactly when the mode had none of the `0222` write bits (`MSDOS_MKATTR`), for files and directories alike. For a hard link, the inode is the hidden data file's.

UMSFUSE keeps the host write permission in step with the record without touching FAT attributes itself: Linux's '`msdos`' and '`vfat`' drivers map `ATTR_RO` to and from the write permission bits, so changing the bits sets the attribute. For every record that is not a directory, a mode without write bits gives the host file '`chmod -w`' (all of `0222` cleared), and a mode with any write bit gives a host file without one '`chmod +w`'; a host file that already has a write bit is left alone, and other bits never change. Pointers and symbolic links have mode `0777`, so their host files stay writable. *UMSFUSE deviation:* directories are left writable on the host. DOS and the Linux FAT drivers ignore the read-only attribute on directories (Linux honours it only with the '`rodir`' mount option), and a host directory without write permission would stop the driver and the tools, run without root privileges, from adding entries to a directory whose mode is, for example, `0555`. `umsfsck` treats a write permission that disagrees with the record as an error.

FAT keeps no execute bits: the Linux FAT drivers give every file the same execute bits from their mount options. The UMSDOS view reports the record's mode and answers `access(2)` from it. On hosts that keep execute bits of their own (tmpfs or ext4, for example), UMSFUSE also keeps the host file's execute bits in step with the record, following the rules for write bits with `0111`, for records that are not directories, symbolic links or pointers; `umsfsck` treats a difference there as something for '`-o`' to update, not as an error. On '`msdos`' and '`vfat`' hosts, execute bits are neither changed nor compared, and a record made from a host file (when a plain directory is promoted, or `umsfsck` adds an unlisted file) gives a regular file no execute bits. This handling has not been tested on a real '`vfat`' mount.

4.5 Timestamps in detail

The `atime`, `mtime` and `ctime` fields are the `time_t` of i386 Linux 2.x: signed 32-bit seconds since 1970-01-01 00:00:00 UTC, in two's complement. They cover 1901-12-13 20:45:52 to 2038-01-19 03:14:07 UTC; `ff ff ff ff` is 1969-12-31 23:59:59 UTC, not a date in 2106. The EMD stores UTC, independent of any time zone.

Linux 2.x sets all three times to the current time when it creates an entry, and updates them as any Unix file system would. Adding or removing a name updates the mtime and ctime of the directory's own record, in its parent's EMD. UMSFUSE does the same.

Regular files are the exception in Linux 2.x: it shows the modification time from the FAT directory entry rather than the record's `mtime`, so a file changed under DOS shows the time of that change (for directories, symbolic links and the other types, the record is used). FAT stores a local time with a 2-second resolution, for the years 1980 to 2107. The record's `mtime` of a regular file is therefore only what Linux 2.x last wrote into it.

UMSFUSE deviation: UMSFUSE takes every time, regular files' modification times included, from the record, and never reads host times except for the tree root and plain directories. Instead, it copies each record's atime and mtime onto its host file (and the directory's own times onto its `--linux-.---`) whenever it changes something, so that Linux

2.x and DOS see the same modification time. On FAT, host times are compared with a 2-second tolerance, and a record time before 1980, which FAT cannot hold, matches a host time at FAT's earliest date. '`umsfsck -o`' repairs host times that differ.

UMSFUSE refuses to store a time outside the 32-bit range when it is given explicitly (`touch -d`, `utime`) with `EOVERFLOW`, leaving the record unchanged. Times that do not come from the caller are clamped instead: the clock, after 2038-01-19 03:14:07 UTC, and the host times of files `umsfsck` adds to an EMD. The '`-o umsclock`' option of `umsdosfs` shifts the clock (see Section 3.1 [umsdosfs], page 10).

4.6 Hard-link records

A FAT file system has no inodes, so UMSDOS represents a file with several names by a hidden data file and one pointer per name.

The *hidden file* is a record with flag `0x01` (and not `0x02`). Its record holds the file's metadata: mode, owner, times and device number, and in `nlink` the number of pointers that refer to it. Its host file holds the data. The hidden file can be of any type except a directory. Its name is `..LINK` followed by the decimal byte offset of its own record, such as `..LINK128` for a record at offset 128; such a name always fits in one slot, and being rejected it gets a mangled host name (`__link12.{_2` for that record). Linux 2.x picks the offset as the first free slot (or the end of the EMD) before it writes the record there. Nothing depends on the name when reading: pointers find the hidden file by its path, and UMSFUSE accepts any name, adding a suffix '`-n`' to a new name only if a hand-made hidden entry already has it.

Each *pointer* is a record with flag `0x02`, mode `0100777` (a regular file with permissions `0777`) and `nlink` 1. Its host file contains the path of the hidden file as text, with no terminating newline or NUL; UMSFUSE ignores trailing newlines and carriage returns when it reads one. The pointer record's owner and times are not the file's: Linux 2.x gives a new pointer the creating process's owner and the current time, `umsdosfs` the owner of the file and the current time. Readers take the metadata, type and data of every name from the hidden file.

To make a link to a plain file, Linux 2.x moves the file's record to a hidden name in the same directory, writes a pointer under the original name and another under the new name, and sets the hidden file's `nlink` to 2. A link to a name that is already a pointer gets a new pointer to the same hidden file, and `nlink` goes up by one. Removing a pointer lowers `nlink`, and removing the last one removes the hidden file.

The hidden file stays in the directory where it was created, even when that directory is renamed or moved, which breaks the paths in pointers outside it (see Section 2.4 [Hard links], page 4). *UMSFUSE deviations:* UMSFUSE puts a new hidden file in the deepest directory that contains all of the file's names, and moves it there (with a new `..LINK` name for its new offset, and every pointer rewritten) when names or directories move. When only one name is left, it turns that name back into a plain file and removes the hidden file, where Linux 2.x keeps a hidden file with a single pointer (see Section 2.4 [Hard links], page 4). Both kinds of tree are valid UMSDOS.

Pointer text

A pointer names its hidden file by a path of UMSDOS names, separated by '/', in one of two formats:

Old format
: A leading '/', then the path from the root of the mounted UMSDOS file system. For a pseudo-root, that root is `\LINUX`, so the path starts below it. Example: `/d/..LINK64`.

New format
: No leading '/', and the path is always taken from the FAT root: on a pseudo-root it starts with `linux/`. Examples: `d/..LINK64`, or `linux/d/..LINK64` for a pseudo-rooted system.

Linux 2.x follows a pointer from the root of the mounted file system (on a pseudo-root, `\LINUX`). The drivers that write the new format skip one leading '/' if there is one, and on a pseudo-root also skip the next 6 bytes, assumed to be `linux/`. The older drivers skip the first byte unconditionally. Each component is looked up like any name, hidden records included. It follows that a driver writing the new format can follow old-format pointers written by a non-root mount, but not those written on a pseudo-root, and that a driver writing the old format cannot follow new-format pointers.

The new format is the one used by the UMSDOS driver's internal version 0.85g, which first appears in Linux 2.2.15 and 2.4.10. That dating matches the Linux 2.2 and 2.4 sources, and has been checked on Slackware systems running kernels from before and after the change.

UMSFUSE writes:

`["/"` in the old format`]` `[`*linkroot* `"/"]` *path*

where *path* is the hidden file's path from the tree root without a leading '/', and *linkroot* is the '`-o umslinkroot`' or '`--link-root`' setting, if any (for example `linux` for a tree that is the FAT's `\LINUX` directory). The new format is the default; '`-o umsoldlinks`' or '`--old-hard-links`' selects the old one. A hidden file at the tree root, without a link root, gets the new-format pointer `..LINK64`, with no '/' at all.

UMSFUSE reads both formats, whatever the settings. It drops one leading '/' and, if a link root is set, the link root and the '/' after it. If the rest does not name a hidden file in the tree, it strips leading components one at a time and tries again, so that a pointer written for a larger tree, such as `/umsdos-tree/..LINK128` in the sample (old format, written by a non-root mount where the tree was the directory `umsdos-tree`), still resolves. A new name added to an existing group copies that group's pointer text. When a hidden file moves, its pointers are rewritten keeping their format and any leading components they had beyond the tree root.

4.7 Special files

The record's file type says how to read its host object. Linux 2.x creates the host object of every type except directories as an ordinary FAT file, and UMSFUSE does the same.

Directories
: The host object is a FAT directory, normally with its own EMD. A record that says directory with a host file that is not, or the reverse, is invalid.

Regular files
: The host file holds the data; the size is the host file's size.

Symbolic links
: Mode `0120777`. The host file contains the link target, with no terminating NUL or newline. As on any Unix file system, the target is an arbitrary string and need not exist.

Character and block devices
: Mode `020000` or `060000` plus permissions, the device number in bytes 20 and 21. The host file is empty. In the sample, `devices/null` is a character device 1, 3 and `devices/hda` a block device 3, 0.

FIFOs and sockets
: Mode `010000` or `0140000` plus permissions. The host file is empty. `umstar` skips sockets, which a tar archive cannot hold.

Hard links
: A pointer is a regular file of mode `0100777` whose host file holds a path (see Section 4.6 [Hard-link records], page 28). The hidden file's record gives the real type, which may be any of the above except a directory.

4.8 Foreign layouts

The record layout follows the sizes of the C types of the architecture the driver was built for. Only the i386 layout is UMSDOS as far as UMSFUSE is concerned: UMSDOS existed to share a FAT partition with DOS and Windows, which means PC hardware. One other layout is known: the one Linux 2.4 writes on PowerPC, and presumably on other ports with 32-bit `uid_t`, `gid_t` and `dev_t`. Its values are still little-endian, but its header is 42 bytes long:

Offset	Size	Field	Contents
0	1	`name_len`	as in i386
1	1	`flags`	as in i386
2	2	`nlink`	as in i386
4	4	`uid`	`00 00`, then the 16-bit value
8	4	`gid`	`00 00`, then the 16-bit value
12	12	`atime`, `mtime`, `ctime`	32-bit each
24	4	`rdev`	`00 00`, minor, major
28	2	`mode`	as in i386
30	12	`spare`	
42	*name_len*	`name`	

A record takes `(42 + name_len + 63) / 64` slots. This layout was seen on a real PowerPC system (Debian 3.1 with Linux 2.4.27).

An EMD is taken to be in this foreign layout when, walking it in the PowerPC layout, there is at least one record and every record is inside the file, has bytes 4, 5, 8, 9, 24 and 25 zero, a valid file type at offset 28 and a usable name at offset 42; and, walking it in the i386 layout, at least one record has an invalid file type at offset 22 or an unusable name at offset 36. An i386 record read as a PowerPC one has its mode in the i386 `spare` bytes, which are zero, so an i386 EMD fails the first test at its first record; a PowerPC record read as an i386 one has its name start in the zeroed `spare` area, so a PowerPC EMD fails the second.

UMSFUSE never reads or writes a directory whose EMD is foreign: `umsdosfs` refuses to mount the tree, even read-only, the read-only tools report an error for that directory, and `umsfsck` reports it as not repairable and skips that directory and everything below it. No other layout has been examined.

4.9 Invariants

A valid UMSDOS tree satisfies the following. The parenthesized notes say what `umsfsck` does when a rule is broken.

1. Every promoted directory has one EMD, a regular file with the host name `--linux-.---` in any case and a size that is a multiple of 64 (warning).
2. Every slot is either free, with all 64 bytes zero, or part of exactly one record, and every record lies entirely inside the file (the record is cleared).
3. Every record has a `name_len` of 1–220 and a name without NUL or '`/`' that is not `.` or `..` (reported). Linux 2.2.26 and 2.4.31 skip a record whose `name_len` is above 220 as a single free slot.
4. The spare bytes, and the bytes after the name in the record's last slot, are zero.
5. No two records in one EMD have the same name (reported for records with the same hidden flag). Since Linux 2.x finds hidden records by name too, a visible and a hidden record should not share a name either.
6. The mode holds a valid file type, and a record has at most one of the two flags; directories have none.
7. The host object named by each record exists in the same host directory (the record is removed), and it is a directory exactly when the record is one (reported).
8. Every host object in a promoted directory, other than the EMD, has a record (a record is added; a valid DOS name that UMSDOS would mangle is renamed to its mangled name).
9. Every pointer's text resolves to a hidden record that is not a directory (the intended target is searched for), and every hidden record's `nlink` equals the number of pointers resolving to it (corrected). A hidden record that no pointer refers to is an error (it is made visible, or an interrupted collapse is finished).

These further rules are UMSFUSE conventions for the host side. They are not required by Linux 2.x:

- Host names are in lower case (a warning; '`-o`' renames them on a case-sensitive host), and every EMD record's host name is the one its current offset gives.
- Each host file's write permission follows its record's mode (an error), and on hosts that keep execute bits so do its execute bits (updated by '`-o`').
- Each host file's modification time equals its record's `mtime` (updated by '`-o`').
- A hard-link group with a single name left is collapsed into a plain file (offered by '`-o`'), and the hidden file of a group lies in the deepest directory containing all of its names, which `umsdosfs` maintains but `umsfsck` does not check.

Appendix A Glossary

8+3
8.3 A two-component file name limit that requires a basename of 1–8 characters, followed by an extension of 0–3 characters.

EMD "Extended MS-DOS Directory:" stored in each directory as the host file `--linux-.---`, this contains all the metadata and file name information to support Unix files on FAT file systems.

host name The file name as it appears on the low-level file system, such as the originally intended FAT basis for UMSDOS.

UMSDOS name
The file name as it is represented in the EMD.

mangling An algorithm to transform Unix names into those compatible with 8+3 and MS-DOS systems. Only very simple names in all-lowercase and within 8+3 limits may get identical host names.