Microsoft

Hardware White Paper

Designing Hardware for Microsoft® Operating Systems

Long Filename Specification

Version 0.5, December 4, 1992
Microsoft Corporation

This document contains the design to support long filenames in the FAT file system.

Contents

Disclaimer

IMPORTANT—READ CAREFULLY: This Microsoft Agreement ("Agreement") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation ("Microsoft") for this version of the Microsoft specification identified above ("Specification"). **BY DOWNLOADING, COPYING OR OTHERWISE USING THE SPECIFICATION, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT DOWNLOAD, COPY, OR USE THE SPECIFICATION.**

The Specification is owned by Microsoft or its suppliers and is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties.

1. **LIMITED COVENANT NOT TO SUE**.

(a) Provided that you comply with all terms and conditions of this Agreement and subject to the limitations in Sections 1(b) – (e) below, Microsoft grants to you the following non-exclusive, worldwide, royalty-free, non-transferable, non-sublicenseable, reciprocal limited covenant not to sue:

(i) under any copyrights owned or licensable by Microsoft without payment of consideration to unaffiliated third parties, to reproduce the Specification solely for the purposes of creating portions of products which comply with the Specification in unmodified form; and

(ii) under its Necessary Claims solely to make, have made, use, import, and directly and indirectly, offer to sell, sell and otherwise distribute and dispose of portions of products which comply with the Specification in unmodified form.

For purposes of the foregoing, the Specification is **"unmodified"** if there are no changes, additions or extensions to the Specification, and **"Necessary Claims"** means claims of a patent or patent application which are (1) owned or licenseable by Microsoft without payment of consideration to an unaffiliated third party; and (2) have an effective filing date on or before December 31, 2010, that must be infringed in order to make a portion(s) of a product that complies with the Specification. Necessary Claims does not include claims relating to semiconductor manufacturing technology or microprocessor circuits or claims not required to be infringed in complying with the Specification (even if in the same patent as Necessary Claims).

(b) The foregoing covenant not to sue shall not extend to any part or function of a product which (i) is not required to comply with the Specification in unmodified form, or (ii) to which there was a commercially reasonable alternative to infringing a Necessary Claim.

(c) The covenant not to sue described above shall be unavailable to you and shall terminate immediately if you or any of your Affiliates (collectively "Covenantee Party") "Initiates" any action for patent infringement against: (x) Microsoft or any of its Affiliates (collectively "Granting Party"), (y) any customers or distributors of the Granting Party, or other recipients of a covenant not to sue with respect to the Specification from the Granting Party ("Covenantees"); or (z) any customers or distributors of Covenantees (all parties identified in (y) and (z) collectively referred to as "Customers"), which action is based on a conformant implementation of the Specification. As used herein, "Affiliate" means any entity which directly or indirectly controls, is controlled by, or is under common control with a party; and control shall mean the power, whether direct or indirect, to direct or cause the direct of the management or policies of any entity whether through the ownership of voting securities, by contract or otherwise. "Initiates" means that a Covenantee Party is the first (as between the Granting Party and the Covenantee Party) to file or institute any legal or administrative claim or action for patent infringement against the Granting Party or any of the Customers. "Initiates" includes any situation in which a Covenantee Party files or initiates a legal or administrative claim or action for patent infringement solely as a counterclaim or equivalent in response to a Granting Party first filing or instituting a legal or administrative patent infringement claim against such Covenantee Party.

(d) The covenant not to sue described above shall not extend to your use of any portion of the Specification for any purpose other than (a) to create portions of an operating system (i) only as necessary to adapt such operating system so that it can directly interact with a firmware implementation of the Extensible Firmware Initiative Specification v. 1.0 ("EFI Specification"); (ii) only as necessary to emulate an implementation of the EFI Specification; and (b) to create firmware, applications, utilities and/or drivers that will be used and/or licensed for only the following purposes: (i) to install, repair and maintain hardware, firmware and portions of operating system software which are utilized in the boot process; (ii) to provide to an operating system runtime services that are specified in the EFI Specification; (iii) to diagnose and correct failures in the hardware, firmware or operating system software; (iv) to query for identification of a computer system (whether by serial numbers, asset tags, user

or otherwise); (v) to perform inventory of a computer system; and (vi) to manufacture, install and setup any hardware, firmware or operating system software.

(e) Microsoft reserves all other rights it may have in the Specification and any intellectual property therein. The furnishing of this document does not give you any covenant not to sue with respect to any other Microsoft patents, trademarks, copyrights or other intellectual property rights; or any license with respect to any Microsoft intellectual property rights.

2. **ADDITIONAL LIMITATIONS AND OBLIGATIONS**.
   (a) The foregoing covenant not to sue is applicable only to the version of the Specification which you are about to download. It does not apply to any additional versions of or extensions to the Specification.
   (b) Without prejudice to any other rights, Microsoft may terminate this Agreement if you fail to comply with the terms and conditions of this Agreement. In such event you must destroy all copies of the Specification.
3. **INTELLECTUAL PROPERTY RIGHTS**. All ownership, title and intellectual property rights in and to the Specification are owned by Microsoft or its suppliers.
4. **U.S. GOVERNMENT RIGHTS**. Any Specification provided to the U.S. Government pursuant to solicitations issued on or after December 1, 1995 is provided with the commercial rights and restrictions described elsewhere herein. Any Specification provided to the U.S. Government pursuant to solicitations issued prior to December 1, 1995 is provided with RESTRICTED RIGHTS as provided for in FAR, 48 CFR 52.227-14 (JUNE 1987) or DFAR, 48 CFR 252.227-7013 (OCT 1988), as applicable.
5. **EXPORT RESTRICTIONS**. Export of the Specification, any part thereof, or any process or service that is the direct product of the Specification (the foregoing collectively referred to as the "Restricted Components") from the United States is regulated by the Export Administration Regulations (EAR, 15 CFR 730-744) of the U.S. Commerce Department, Bureau of Export Administration ("BXA"). You agree to comply with the EAR in the export or re-export of the Restricted Components (i) to any country to which the U.S. has embargoed or restricted the export of goods or services, which currently include, but are not necessarily limited to Cuba, Iran, Iraq, Libya, North Korea, Sudan, Syria and the Federal Republic of Yugoslavia (including Serbia, but not Montenegro), or to any national of any such country, wherever located, who intends to transmit or transport the Restricted Components back to such country; (ii) to any person or entity who you know or have reason to know will utilize the Restricted Components in the design, development or production of nuclear, chemical or biological weapons; or (iii) to any person or entity who has been prohibited from participating in U.S. export transactions by any federal agency of the U.S. government. You warrant and represent that neither the BXA nor any other U.S. federal agency has suspended, revoked or denied your export privileges. For additional information see http://www.microsoft.com/exporting.
6. **DISCLAIMER OF WARRANTIES. To the maximum extent permitted by applicable law, Microsoft and its suppliers provide the Specification (and all intellectual property therein) and any (if any) support services related to the Specification ("Support Services")** ***AS IS AND WITH ALL FAULTS*****, and hereby disclaim all warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties or conditions of merchantability, of fitness for a particular purpose, of lack of viruses, of accuracy or completeness of responses, of results, and of lack of negligence or lack of workmanlike effort, all with regard to the Specification, any intellectual property therein and the provision of or failure to provide Support Services. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT, WITH REGARD TO THE SPECIFICATION AND ANY INTELLECTUAL PROPERTY THEREIN. THE ENTIRE RISK AS TO THE QUALITY OF OR ARISING OUT OF USE OR PERFORMANCE OF THE SPECIFICATION, ANY INTELLECTUAL PROPERTY THEREIN, AND SUPPORT SERVICES, IF ANY, REMAINS WITH YOU.**
7. **EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SPECIFICATION, ANY INTELLECTUAL PROPERTY THEREIN, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS AGREEMENT, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**
8. **LIMITATION OF LIABILITY AND REMEDIES. Notwithstanding any damages that you might incur for any reason whatsoever (including, without limitation, all damages referenced above and all direct or general damages), the entire liability of Microsoft and any of its suppliers under any provision of this Agreement and your exclusive remedy for all of the foregoing shall be limited to the greater**

of the amount actually paid by you for the Specification or U.S.$5.00. The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails its essential purpose.

9. **APPLICABLE LAW**. If you acquired this Specification in the United States, this Agreement is governed by the laws of the State of Washington. If you acquired this Specification in Canada, unless expressly prohibited by local law, this Agreement is governed by the laws in force in the Province of Ontario, Canada; and, in respect of any dispute which may arise hereunder, you consent to the jurisdiction of the federal and provincial courts sitting in Toronto, Ontario. If this Specification was acquired outside the United States, then local law may apply.
10. **QUESTIONS**. Should you have any questions concerning this Agreement, or if you desire to contact Microsoft for any reason, please contact the Microsoft subsidiary serving your country, or write: Microsoft Sales Information Center/One Microsoft Way/Redmond, WA 98052-6399.
11. **ENTIRE AGREEMENT. This Agreement is the entire agreement between you and Microsoft relating to the Specification and the Support Services (if any) and they supersede all prior or contemporaneous oral or written communications, proposals and representations with respect to the Specification or any other subject matter covered by this Agreement. To the extent the terms of any Microsoft policies or programs for Support Services conflict with the terms of this Agreement, the terms of this Agreement shall control.**

Si vous avez acquis votre produit Microsoft au CANADA, la garantie limitée suivante vous concerne :

RENONCIATION AUX GARANTIES. Dans toute la mesure permise par la législation en vigueur, Microsoft et ses fournisseurs fournissent la Specification (et à toute propriété intellectuelle dans celle-ci) et tous (selon le cas) les services d'assistance liés à la Specification ("Services d'assistance") TELS QUELS ET AVEC TOUS LEURS DÉFAUTS, et par les présentes excluent toute garantie ou condition, expresse ou implicite, légale ou conventionnelle, écrite ou verbale, y compris, mais sans limitation, toute (selon le cas) garantie ou condition implicite ou légale de qualité marchande, de conformité à un usage particulier, d'absence de virus, d'exactitude et d'intégralité des réponses, de résultats, d'efforts techniques et professionnels et d'absence de négligence, le tout relativement à la Specification, à toute propriété intellectuelle dans celle-ci et à la prestation ou à la non-prestation des Services d'assistance. DE PLUS, IL N'Y A AUCUNE GARANTIE ET CONDITION DE TITRE, DE JOUISSANCE PAISIBLE, DE POSSESSION PAISIBLE, DE SIMILARITÉ À LA DESCRIPTION ET D'ABSENCE DE CONTREFAÇON RELATIVEMENT À LA SPÉCIFICATION ET À TOUTE PROPRIÉTÉ INTELLECTUELLE DANS CELLE-CI. VOUS SUPPORTEZ TOUS LES RISQUES DÉCOULANT DE L'UTILISATION ET DE LA PERFORMANCE DE LA SPÉCIFICATION ET DE TOUTE PROPRIÉTÉ INTELLECTUELLE DANS CELLE-CI ET CEUX DÉCOULANT DES SERVICES D'ASSISTANCE (S'IL Y A LIEU).

EXCLUSION DES DOMMAGES INDIRECTS, ACCESSOIRES ET AUTRES. Dans toute la mesure permise par la législation en vigueur, Microsoft et ses fournisseurs ne sont en aucun cas responsables de tout dommage spécial, indirect, accessoire, moral ou exemplaire quel qu'il soit (y compris, mais sans limitation, les dommages entraînés par la perte de bénéfices ou la perte d'information confidentielle ou autre, l'interruption des affaires, les préjudices corporels, la perte de confidentialité, le défaut de remplir toute obligation y compris les obligations de bonne foi et de diligence raisonnable, la négligence et toute autre perte pécuniaire ou autre perte de quelque nature que ce soit) découlant de, ou de toute autre manière lié à, l'utilisation ou l'impossibilité d'utiliser la Spécification, toute propriété intellectuelle dans celle-ci, la prestation ou la non-prestation des Services d'assistance ou autrement en vertu de ou relativement à toute disposition de cette convention, que ce soit en cas de faute, de délit (y compris la négligence), de responsabilité stricte, de manquement à un contrat ou de manquement à une garantie de Microsoft ou de l'un de ses fournisseurs, et ce, même si Microsoft ou l'un de ses fournisseurs a été avisé de la possibilité de tels dommages.

LIMITATION DE RESPONSABILITÉ ET RECOURS. Malgré tout dommage que vous pourriez encourir pour quelque raison que ce soit (y compris, mais sans limitation, tous les dommages mentionnés ci-dessus et tous les dommages directs et généraux), la seule responsabilité de Microsoft et de ses fournisseurs en vertu de toute disposition de cette convention et votre unique recours en regard de tout ce qui précède sont limités au plus élevé des montants suivants: soit (a) le montant que vous avez payé pour la Spécification, soit (b) un montant équivalant à cinq dollars U.S. (5,00 $ U.S.). Les limitations, exclusions et renonciations ci-dessus s'appliquent dans toute la mesure permise par la législation en vigueur, et ce même si leur application a pour effet de priver un recours de son essence.

DROITS LIMITÉS DU GOUVERNEMENT AMÉRICAIN

Tout Produit Logiciel fourni au gouvernement américain conformément à des demandes émises le ou après le 1er décembre 1995 est offert avec les restrictions et droits commerciaux décrits ailleurs dans la présente convention. Tout Produit Logiciel fourni au gouvernement américain conformément à des demandes émises avant le 1er décembre 1995 est offert avec des DROITS LIMITÉS tels que prévus dans le FAR, 48CFR 52.227-14 (juin 1987) ou dans le FAR, 48CFR 252.227-7013 (octobre 1988), tels qu'applicables.

Sauf lorsqu'expressément prohibé par la législation locale, la présente convention est régie par les lois en vigueur dans la province d'Ontario, Canada. Pour tout différend qui pourrait découler des présentes, vous acceptez la compétence des tribunaux fédéraux et provinciaux siégeant à Toronto, Ontario.

Si vous avez des questions concernant cette convention ou si vous désirez communiquer avec Microsoft pour quelque raison que ce soit, veuillez contacter la succursale Microsoft desservant votre pays, ou écrire à: Microsoft Sales Information Center, One Microsoft Way, Redmond, Washington 98052-6399.

1. Overview

This document contains the design to support long filenames in the FAT file system.

2. Problem Description and Objectives

The length supported for a long filename must be a minimum of 32 bytes. That is each component of a long path name may contain at least 32 character. The maximum file path name is limited to 260 characters. This is the limit imposed for single byte character path names on Windows NT®. Defining a reasonable maximum limit to the filename length makes application writing easier. This way the application knows the maximum size of a buffer that is required to hold the entire file path name.

Long file names will be stored with the case preserved; they will not be translated into upper case as 8.3 file names are. The case of a long name will not be matched on finds, opens, creates, etc., the match will be case insensitive. The case in a long name, which is stored when it is created, will be returned on find operations. Therefore if a file is created with the name "Foo", then later searched for through a find operation with the name "FOO", the same file will be found. If a long-name find operation is used, the file name "Foo" will be returned. This also means that the long names "Foo" and "fOO" may not exist in the same directory.

The name space between long and 8.3 names will be common. In other words a long name for a file cannot be the same as a short name in another file. If a file is created with a short name API the long name is forced to be the same. If there is a conflict it will always occur in the short name space. For example, if the file "FOO" is created by a short name API, the long name will be forced to be "FOO". To insure this can take place, a file created with a long name API that is given a valid 8.3 name, the short name must be forced to be the same. For example, if the file "Foo" is created by a long name API, the short name will be forced to be "FOO". If there is a conflict in either the long or short name space then an error must be returned.

The long name APIs will operate uniformly on both long and short names. An open of a file, through a long name API, does not match case and will open either a long or short name which ever exists. The 8.3 APIs will only operate on the short name space. The long name space is not visible to these APIs.

Logically both a long and a short name will always be created for a file. In practice if a long name API creates a file that is a valid 8.3 name, only an 8.3 entry may be created for the name. If the name is longer than 8.3, or contains lower case characters, then a long name entry must be created for the file and an 8.3 name will automatically be generated from it and stored in the 8.3 entry. If a long name is renamed, the 8.3 name stored for it will by default be automatically recreated. The application will not be allowed to define, or independently change the short name that is automatically created. If an existing Int 21h rename function is called to rename an 8.3 filename that also has a long name, the long name will be changed as well to the 8.3 name. This may actually be stored only in the 8.3 entry.

The valid character set for long file names includes the current set of valid file name characters for the FAT based 8.3 file system plus a few additional characters. There are no characters removed from the valid file name character set defined for MS-DOS® operating system versions >= 3.00. The biggest difference, in terms of valid filename characters, is that LOWER CASE characters are allowed in the name stored in the directory entry. Note also that SPACE (20h) is a valid file name character, and note that this is not a change from the existing 8.3 MS-DOS valid file name character convention.

The current directory is limited to 64 characters based on the 8.3 names. The long name for the current directory may be much longer, up to 260 bytes, however setting the current directory through a

long name API can not result in the 8.3 current directory exceeding 64 characters. This is required for compatibility with the existing current directory limitations.

The handling of long names in the Microsoft® Windows® operating system must be consistent with the NTFS and OFS file systems. Both of these file systems support long and short names, and are fully compatible with each other. Any solution provided on FAT must be compatible with these environments as well.

The long name APIs for MS-DOS and WIN16 applications will be multiplexed through a single Int 21h function.

It is desirable to keep the existing FAT format compatible so that long name media may be interchanged with a down level system.

3. Solution and Justification

The original idea for supporting long filename was to have a separate hidden file in each directory that contains the long names. This file would have to contain both long and 8.3 names. There are a number of problems with this idea related to keeping this long name directory file in sync. As a result it has been discarded in favor of the preferred solution defined below.

3.1. Proposed Solution

One idea has been proposed to use additional directory entries in addition to the existing 8.3 entry to contain the long name. This idea adds additional 32 byte directory entries adjacent to the existing directory entry. The existing directory entry would be unmodified to provide the 8.3 name compatibility.

The initial idea was to setup the additional entries to look like unused (deleted) 32 byte directory entries. This approach has since been changed to use valid file entries with the attribute byte set to HIDDEN, SYSTEM, READ-ONLY, and VOLUME LABEL. Each additional directory entry also contains a signature byte and a byte checksum, or CRC, of the short 8.3 name. The First Cluster field of these entries is also set to 0.

The setting of the volume label makes these entries almost invisible on down level systems, and prevents them from being accidentally used. The TRUE 8.3 volume label directory entry has only the VOLUME LABEL bit set in the attribute byte. This allows the LFNFAT file system to be able to easily distinguish the true volume label entry and removes any order restriction on the true volume label . The true volume label entry can be anywhere in the root directory, it doesn't have to be first, but we would like it to be so that down level systems will find the correct label.

The signature byte and the attribute byte form a validation signature that identifies the entry as a long name for the associated file. The signature byte contains an ordered value for each additional directory entry (each entry, first to last, has a unique signature value).

The byte following the attribute byte contains a check sum, or CRC of the 8.3 file name. This check sum is used to help insure the long name is valid for the associated 8.3 directory entry. This is useful when a disk is taken to a down level system and the 8.3 names are changed.

The "first cluster" field of the long name directory entry is restricted to always be zero. This prevents existing tools that directly access the disk from thinking that clusters are cross-linked by these long name directory entries. With out this data could be lost by these tools trying to "clean up" the disk.

The remaining bytes in the long name directory entries are used to contain long file name characters. Since the long name directory entries are not visible files on down level systems, any restrictions on using the 10 unused 32-byte directory entry bytes can probably be relaxed. Each long name entry then contains 27 available bytes. With the setting of the attribute byte, checksum byte, signature byte, and zeroing the first_cluster field in the long name entries, these available bytes are not contiguous. Instead they are separated into three disjoint pieces.

Originally we were considering using a fixed number of additional entries to support long names greater than 32 characters, (the Mac file name size). However, it is desirable to handle long names compatibly with the NTFS and Win32® API, which support up to 255-character file names. To support file names up to 255 characters long, the number of long name directory entries used will be dynamically allocated. Only the number of entries required to hold the long name will be used. If the long name is 27 characters or less, a single entry will be used. If the long name is more than 27 characters, multiple entries will be used, up to a maximum of 255 characters. The only difference in the format of multiple long name entries will be in the signature byte. The signature byte is an ordered value for each entry in a given long name. Using ordered signature values per entry simply helps reduce the possibility of invalid detection of a long name directory. This way, by looking at the signature in the long name entry, the position of the entry within the long name can be identified.

If the long name does not fill the entire entry, then a null terminator will be placed in the string so that additional spaces, or garbage bytes will not be returned. The long name directory entries will not be padded with spaces as the 8.3 directory entries are. When a long name directory entry is allocated, the parts of it reserved for name characters are filled with 0FFh. When a long name is shortened, through rename, the extra bytes in the last extension past the new position of the terminating null are 0FFh-filled. The reason for this is an artifact of the behavior of NDD and DISKFIX, which will allow the Windows disk repair code to detect that one of these applications has modified the entry. This is an easy addition to the rename/create code and is of low performance impact.

The additional entries immediately precede the existing 8.3 directory entry. By placing the long name entries prior to the 8.3 directory entry they may be cached while searching for a valid 8.3 directory entry. Once a valid entry is found the long name entries will already be available in memory. This simplifies the code to implement long file names by preventing the need to handle boundary conditions while searching for long names at the end of the directory.

There is no reason that this directory extension architecture cannot be applied to the Volume Label as well as directory and file names. This allows volume labels to have up to 255 characters, the same as file names.

The use of additional directory entries has the advantage that it keeps the long name associated with the true directory entry. This makes keeping the long and short names in sync much easier. The search for a filename would be more localized on the disk, which would make it faster. It would not require as much disk space due to a single copy of the 8.3 name and a single cluster chain being allocated, rather than two cluster chains for the hidden file mechanism.

The only other problems that have been identified are performance concerns with extracting the long filenames. The long name signature needs to be checked, and the short name needs to be checksumed to validate the long name. The long name itself may also be discontiguous if it spans 2 directory entries. It is also possible for a long name to spans across a disk sector boundary. Despite these performance concerns the impact is not as bad as implementing long names through a hidden file.

There are also remote possibilities that a deleted non-long name directory entry contains a valid signature, which would be misdetected as a long name entry. The checksum of the 8.3 name also has the possibility of matching when it should not. These are both possible cases that could exist,

however they are very rare and would simply result in a long name incorrectly being associated with an 8.3 directory entry.

This architecture allows both new and old format directory entries to be intermixed in the same directory. It would be nice NOT to support this intermixing for performance and code complexity reasons, but this is required anyway to allow support of old disk utilities and changes made to the disks by down level systems. It also saves space in the directory entry. This is important in the root directory since it is fixed in length and cannot be dynamically grown.

3.2. Automatic Name Generation

The name space used for long and short names is considered to be a single name space such that conflicts will not occur between long and short names. In other words, a long name for a file may not contain the same name, ignoring case, as the short name in a different file. This prevents confusion for the user regarding the name of a file. To handle this, when ever a file name is created or changed the alternate name is automatically generated for the file. If a long name API is used, then the short name will be automatically generated. If an existing short name API is used, then the long name will be automatically generated.

3.2.1. The basic generation algorithm

The basic generation algorithm has two parts. The first part is to generate a starting name or basis, and the second part is to modify the basis until the name is unique.

For example, if the user specified a long name of "LONGNAMEFILE.EXE" the closest corresponding 8.3 name will be generated with the numeric value "-1" added to use as a basis. In this example the 8.3 basis of the long name is "LONGNA-1.EXE". If the file name "LONGNA-1.EXE" already exists in the directory, then the second part of the algorithm is used to modify the name until it is unique. In this example, "LONGNA-1.EXE" will be changed to "LONGNA-2.EXE", and if that is not unique then "LONGNA-3.EXE", and so forth.

The basis name that is tried always contains the numeric value, starting with "-1". This is done to prevent auto-generated names from using up the more commonly used 8.3 name space. This also makes it somewhat easier to identify auto generated 8.3 names.

3.2.2. File name extensions

An important goal in the automatic name generation strategy is to preserve the file extensions as much as possible. For example, if a long file name ends in ".EXE" then the 8.3 name will also end in ".EXE". Note that long names can have multiple extensions (defined by having multiple periods in the file name), but even then an attempt will be made to make some sense out of the extension.

3.2.3. Generating long names from 8.3 names

When an 8.3 name API creates a file, the long name is always defined as being the 8.3 name. If a conflict exists an error will be returned and the creation of the file will fail. Note this will only occur if an 8.3 name already exists with the same name. Existing applications will always be able to locate the file that is in conflict.

3.2.4. Generating 8.3 names from long names

When a long name API creates a file, the long name is first checked to see if it conforms to a valid 8.3 format, that is, ignoring case. If it does conform, then the long name is defined as the 8.3 name. If a conflict exists an error will be returned and the creation of the file will fail. This allows the creation of an 8.3 name to always be able to create a long name that is the same as the 8.3 name. Note that the file name causing the long name API to fail may be an 8.3 file name.

The following steps are used to form an 8.3 basis from a long name.

1. Remove all spaces. For example "My File" becomes "MyFile".

2. Initial periods, trailing periods, and extra periods prior to the last embedded period are removed. For example ".logon" becomes "logon", "junk.c.o" becomes "junkc.o", and "main." becomes "main".

3. Translate all illegal 8.3 characters into "_" (underscore). For example, "The[First]Folder" becomes "The_First_Folder".

4. If the name does not contain an extension then truncate it to 6 characters. If the names does contain an extension, then truncate the first part to 6 characters and the extension to 3 characters. For example, "I Am A Dingbat" becomes "IAmADi" and not "IAmADing.bat", and "Super Duper Editor.Exe" becomes "SuperD.Exe".

5. If the name does not contain an extension then a "-1" is appended to the name. If the name does contain an extension, then a "-1" is appended to the first part of the name, prior to the extension. For example, "MyFile" becomes "MyFile-1, and "Junk.bat" becomes "Junk-1.bat". This numeric value is always added to help reduce the conflicts in the 8.3 name space for automatic generated names.

6. This step is optional dependent on the name colliding with an existing file. To resolve collisions the decimal number, that was set to 1, is incremented until the name is unique. The number of characters needed to store the number will grow as necessary from 1 digit to 2 digits and so on. If the length of the basis (ignoring the extension) plus the dash and number exceeds 8 characters then the length of the basis is shortened until the new name fits in 8 characters. For example, if "FILENA-1.EXE" conflicts the next names tried are "FILENA-2.EXE", "FILENA-3.EXE", ..., "FILEN-10.EXE", "FILEN-11.EXE", etc.

3.2.5. Handling international character sets

The long name character set will always be ANSI. This stays compatible with windows applications where everything is ANSI. In the future this will be expanded to Unicode, which is a super set of ANSI. The short name character set will always be OEM. This stays compatible with MS-DOS and existing windows applications that store 8.3 file names using the OEM character set.

The problem with the OEM code page is when it converts from lower to upper case it maps multiple extended characters to a single upper case character. This creates problems because it does not preserve the information that the extended character provides. It also prevents the creation of some file names that are different, but because of the mapping to upper case they become the same file name.

The ANSI code page solves this problem by always providing a translation for lower case characters to a single and separate upper case character. It would be nice to use ANSI for both long and short names, but this creates problems with MS-DOS applications, which expect the short names to be in the OEM code page. If we used ANSI for short file names, the MS-DOS application would no longer be able to properly display many of the characters in these file names since the OEM code page does not contain a number of the upper case characters that exist in ANSI. This means file names containing these ANSI characters would be displayed as some strange OEM code page symbol.

The mapping of short names to long names is not a problem for normal text characters. Since all upper case OEM characters can be mapped to the proper character in the ANSI character set, the short names can always be mapped and displayed using the ANSI character set. If extended graphics characters are used in the short file name then the mapping will not be as consistent. These characters are not commonly used in file names and no special handling is provided for them.

The automatic generation of short names from long names creates more problems. Conflicts can occur when different long names that fit within the 8.3 format contain extended characters. To prevent these conflicts in the mapping of long ANSI names to short OEM names the -number (-1,-2,...) will be added to a short filename when the long name contains an extended character that maps to a character below 128.

To make this more generic, if the long name does not map directly to the short name such that each character has a valid upper case translation, then the -number will be appended to the short name. This addition of the -number takes place even if the name would otherwise fit within a valid 8.3 format. This helps prevent collisions in the short name space as a result of the mapping. This includes the mapping of invalid characters to an underscore, which will result in the same type of conflict.

This use of the -number is consistent with the handling for long names that do not fit in the 8.3 space. The use of extended characters simply forces the generation of a short name to follow the same rules as names that don't fit in the 8.3 space, even if this would otherwise not be the case.

The mapping of the extended characters follows the same process that is uses today in Windows 3.1 to map ANSI to OEM. This remains consistent with what Windows has done in the past.

This allows long names to preserve the extended character information, and always map to a non-conflicting short name that can be displayed using the OEM character set.

Note this is not consistent with the NTFS solution, which maps all extended characters to an underscore in the short name. The NTFS file system handles the mapping of extended characters from long to short names by treating any character greater than 127 as an invalid character. The creation of a short name converts all invalid characters in to an underscore '_'. This prevents them from having any knowledge of code pages in the translation of characters. It also does not solve the problem caused by mapping multiple characters into a single character.

3.3. Search Operations

For short name APIs, the passed in name is in the OEM character set. This name is converted to upper case before it is used in a search operation in the short name space. The long name space is not searched for short name APIs.

For long name APIs, the passed in name is in the ANSI character set. A long name API search operation must cover both the long and short name spaces. The long-name find API will return both the long and short name in the result buffer. The returned long name will be in the ANSI character set and the short name will be in the OEM character set.

To handle a long name search for a file with extended characters, two conversions will take place on the passed in name. For searching the long name entries the name will be converted to upper case using the ANSI character set. Each entry long name directory entry will also be converted to upper case for the comparison. For searching the short name entries the name will first be converted to upper case using the ANSI character set, then it will be converted from ANSI to the OEM character set. If the conversion to the OEM character set results in a mapping of an extended character to a character below 128, then the short names will not be searched. This is because the name being searched for cannot exist in the OEM character set.

For short file names that are stored as only a short name and containing extended characters, the long name find API will convert the short name to ANSI and return the result as the long name for the file. This conversion from OEM to ANSI will be the same as that done in Windows 3.1.

3.4. MS-DOS APPLICATION Support of Long Names

MS-DOS applications will not be prevented from using the long name APIs. However MS-DOS applications may not be able to properly display a long file name. This occurs because long file names use the ANSI character set and may contain ANSI characters that cannot be properly displayed using the OEM character set. The MS-DOS applications must be aware of this and deal with these ANSI file names as they see appropriate.

This is also a reason that we should not provide long name support in the MS-DOS command line utilities that we provide with the system.

3.5. Effect of Down Level Systems

The support of long file names is most important on the hard disk, however it will be supported on removable media as well. The proposed architecture provides support for long names without breaking compatibility with the existing FAT format. A disk can be read by a down level system with out any compatibility problems. An existing disk does not go through a conversion process before it can start using long names. All of the current files remain unmodified. The long name directory entries are added when a long name is created. The addition of a long name to an existing file may require the 8.3 directory entry to be moved if the required adjacent directory entries are not available.

The long name entries are as hidden as hidden or system files are on a down level system. This is enough to keep the casual user from causing problems. The user can copy the files off using the 8.3 name, and put new files on without any side effects.

The interesting part of this is what happens when the disk is taken to a down level MS-DOS system and the directory is changed. This can affect the long name entries since the down level system ignores these long names and will not insure they are properly associated with the 8.3 name.

A down level system will only see the long name entries when searching for a label. On a down level system, the volume label will be incorrectly reported if the true volume label does not come before all of the long name entries in the root directory. This is because the long name entries also have the volume label bit set. This is unfortunate, but not considered a critical enough problem considering the alternatives.

If an attempt is made to remove the volume label, one of the long name directory entries may be deleted. This would be a rare occurrence, but should be easy to detect on the Windows system, the long name entry will not be a valid file entry, since it will be marked as deleted. If the deleted entry is reused, then the attribute byte will not have the proper value for a long name entry. The down side to this is a subsequent attempt to create a volume label will fail if the true volume label still exists. The label command could run into this when it renames a volume label, which would be confusing to the user.

If a file is renamed on a down level system, then only the short name will be renamed. The long name will not be affected. Since the long and short names must be kept consistent across the name space, it is desirable to have the long name become invalid as a result of this rename. The checksum, or CRC, of the 8.3 name that is kept in the long name directory provides the ability to detect this type of change. This checksum will be checked to validate the long name before it is used. Rename will cause problems only if the renamed 8.3 file name happens to have the same checksum. We need to carefully evaluate the checksum method we use. The "duplicate checksum" frequency is critical to how frequent this bad behavior could be.

This rename of the 8.3 name must also not conflict with any of the long names. Otherwise a down level system could create a short name in one file that matches a long name, when case is ignored, in a different file. To prevent this, the automatic creation of an 8.3 name from a long name that follows

the 8.3 format will directly maps the long name to the 8.3 name by converting the characters to upper case.

If the file is deleted, then the long name is simply abandon. If a new file is created, the long name may be incorrectly associated with the new file name. As in the case of a rename the checksum of the 8.3 name will help prevent this incorrect association.

3.6. Affect on Existing Disk Utilities

Chkdsk on a down level system does not complain about these long names on a disk and finds nothing wrong with them.

The following are some evaluations that have been done using PC-TOOLS and NORTON to understand the effect they have on a disk containing long file names.

SPEEDDISK sorts directories without paying any attention to the "extra volume labels". The fact that this moves the LFN entries is the only problem with this. This is the reason for the signature bytes in the extensions to identify the extension order so that the directory can be automatically fixed by Windows chkdsk in all cases where there are not checksum collisions in the directory. Without the ordering of the signatures, the USER will have to be asked to group the extensions in the proper order for all files with more than one LFN extension.

COMPRESS directory sort seems to be totally disabled by this. It compresses the disk and leaves the directories completely alone. Thus running COMPRESS on an LFN disk seems to have no side effects.

NDD does not like the file size field being non-0 when the first cluster pointer is 0 and wants to overwrite the file size field with 0. To handle this the unused bytes of the extension entries are non-zero padded, this allows this mod to be detected by the Windows chkdsk and the user asked to help fix it. This is the only thing that NDD seems to be upset about.

DISKFIX has the same behavior as NDD regarding the file size field, plus it gets very upset when certain of the 10 reserved bytes are non zero. In the case where one of the 10 reserved bytes it cares about has something non-zero in it, it COMPLETELY erases the entry. It writes E5 in the first byte, and 0 in all other bytes. For this reason, the check sum byte is placed in this reserved area. This way the behavior of DISKFIX will become more predictable, it will totally erase all LFN extension entries. This is bad, but its going to do something bad no matter how this is handled. It is preferable to have a predictable bad behavior than an unpredictable one.

CPS MS-DOS Anti-Virus and McAfee SCAN do not think these disks are infected.

None of these applications seems to care about the extensions having MS-DOS 5.00 illegal file name characters in them. The invalid characters are totally ignored. UNDELETE may have a different opinion.

OTHER INTERESTING INFORMATION COLLECTED:

Set the directory bit is bad. Downlevel CHKDSK reports all the extensions as invalid sub directories. NDD is upset and turns off some of the DIR attribute bits (not all of them interestingly?!?!?!?!?). DISKFIX will erase all of the LFN extensions regardless of whether the 10 reserved bytes are non-zero. COMPRESS bails: Insufficient memory. SPEEDDISK unaffected. Not setting the VOLUME LABEL bit is bad. Downlevel CHKDSK will find the extension files and complain about the fact that the file size field is wrong and want to 0 it.

Non zero values in First Cluster is very bad. If what is in there happens to look like a cross link, very evil things start happening. NDD got very upset and changed one of the 8.3 entries to 0 size (because it was cross linked), the data became "lost" and was assigned to an NDD "lost data" file. SPEEDDISK and COMPRESS hung, or reported disk trashed and bailed. CHKDSK and DISKFIX were unaffected.

3.7. Network and Other File System Support

Long file names will be supported in the protected mode FAT file systems in Windows. However, other file systems may be accessed by Windows that might not support long file names. The most common case will be network resources. There are three cases that must be handled here, systems with 8.3 name only, systems with long names only, and system with both long and 8.3 names.

A file system that only supports 8.3 names, such as CD-ROM, handles the 8.3 name APIs with out an issue. When a long name API is used to access a file, the API will be mapped into its corresponding 8.3 API. This mapping process will validate the file name to insure that it is a valid 8.3 name. If the name is longer than 8.3, an error will be returned to the caller. The long name will not be passed to the file system since it would truncate the name and not return an error. The IFSMGR must know if a file system driver handles long name or not in order to provide this support. The IFSMGR is responsible for making sure that long names are never passed to a file system that cannot handle them. This may require a new "Get FSD Info" function to provide this information to a long name aware application. This will allow long name aware applications to adjust their behavior when running on non long name file systems.

The HPFS file system is an example of a file system that only supports long name. The results of an 8.3 name API request to a long name only file system will depend on the particular file system. For HPFS all valid 8.3 file names will be handled as expected. These APIs will be able to access all of the files that have a valid 8.3 file name. The long names, those that do not fit in the 8.3 format, will not be visible through these APIs. The long name APIs will be able to access all files on the file system. The one difference will be in the handling of the long name find API. The Win32 find API returns both the long and 8.3 name for a file. On a long name only file system the 8.3 file name is not defined, therefore the Win32 find API must allow the returned short name to be undefined. This is not a change in the Win32 find API return structure, it simply means that a null string may be returned for the short name.

File systems that support both long and 8.3 file names are not an issue, except for networked devices. The current LANMAN SMB protocol does not support returning both long and 8.3 names on a find operation. This is required for the Win32 find API. To support this the transact2 SMB protocol will be used with a new function defined.

3.8. FAT File Name Character Set

A valid FAT file system file names has the following form:

- [\][directory\]filename[.extension]

The directory parameter specifies the directory that contains the file's directory entry. The directory may be made up of multiple components, each being separated by a backslash (\). The last directory component must be followed by a backslash (\) to separate it from the filename. There are two special directory components, (.) and (..). The director component (.) represents the current directory. The directory component (..) represents the directory one level up (closer to the root). If the specified directory is not in the current directory, the directory must include the names of all directories in the path, separated by backslashes. The root directory is specified by using a backslash at the beginning of the name. For example, if the directory "abc" is in the directory "sample" and "sample" is in the root directory, the correct directory specification is "\sample\abc".

In an 8.3 file name, the directory name and file name consist of up to 8 characters followed by an optional period (.) and extension of up to 3 characters. The characters may be any combination of letters, digits, or the following special characters:

- $ % ' - _ @ ~ ` ! ( )

In a long file name, the directory and file name components consist of up to 48 characters. The characters may be any combination of those defined for 8.3 names with the addition of the period (.) character.

The following 7 characters : + , ; = [ ] are legal in a long name but illegal in 8.3 names. A space is also a valid character in a long name, it always has been for 8.3 name also however it just does not get used.

4. Data Structures Description

The layout of a long name directory entry appears as follows.

Last Long Name Directory Entry
. . .
2nd Long Name Directory Entry
1st Long Name Directory Entry
Existing 8.3 Directory Entry

Note that the number of long name directory entry will depend on the length of the long name.

4.1. Long Name Directory Entry Structure

```
LNDIRENT STRUCT
        dir_lname1      DB      10 DUP (?)  ; long name string
        dir_sig DB      ?               ; signature byte
        dir_attr        DB      ?               ; file attributes
        dir_flags       DB      ?               ; flags byte (TBD)
        dir_chksum      DB      ?               ; checksum of 8.3 name
        dir_lname2      DB      12 DUP (?)  ; long name string
        dir_first       DW      ?               ; first cluster number, must be 0
        dir_lname3      DB      4 DUP (?)   ; long name string
LNDIRENT ENDS
```

5. Design Overview

TBD.

6. Exported Interfaces

The following APIs need to be provided to support long filenames.

Int 21 file attributes function
Int 21 file delete function
Int 21 file dir function (make, remove, change, get)
Int 21 file find function
Int 21 file open/create function
Int 21 file rename function

These APIs will be supported through a single Int 21h function. The original idea was to pass the parameters in a structure. This has been changed to look just like the current Int 21h interfaces, with

the existing function number placed in AL. These functions are the same as their existing counterpart, with the exception that they will accept and return long names as appropriate.

The Find APIs are an exception. The format of the search result buffer is changed to follow the Win32 Find API result buffer. The Find APIs also return and uses a handle, the same as the Win32 APIs, to identify the search that is in progress.

This solution greatly simplifies the work required to support these functions in the system. The long name functions that are passed on to existing file system can be easily mapped into the 8.3 APIs. The translation of these parameters in to the protected mode file system interface already exists for the 8.3 APIs. With this interface the long-name APIs can be handled with the same mapping code.

6.1. Long Name Int 21h Function

```
;**     longnameAPI - Int 21h function to support long name APIs
;
;       For Get/Set Attributes,
;
;       Entry   AH = LFN_FUNC
;               AL = 043h
;               BL = 000h - if getting file attributes
;                  = 001h - if setting file attributes
;               CX = new file attributes, if BL = 001h
;               DS:DX = file pathname
;       Exit    If successful, carry flag is clear
;                   CX = returns file attributes
;               else, carry flag is set
;                   AX = error code
;
;       For Delete File,
;
;       Entry   AH = LFN_FUNC
;               AL = 041h
;               DS:DX = file pathname
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
;
;       For Change Directory,
;
;       Entry   AH = LFN_FUNC
;               AL = 03Bh
;               DS:DX = directory pathname to change to
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
;
;       For Get Current Directory,
;
;       Entry   AH = LFN_FUNC
;               AL = 047h
;               DL = drive code
;               DS:SI = 260 byte buffer to fill in
;       Exit    If successful, carry flag is clear
;                   buffer filled in with current directory pathname
;               else, carry flag is set
;                   AX = error code
;
;       For Make Directory,
;
;       Entry   AH = LFN_FUNC
;               AL = 039h
;               DS:DX = directory pathname to make
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
```

```
;
;       For Remove Directory,
;
;       Entry   AH = LFN_FUNC
;               AL = 03Ah
;               DS:DX = directory pathname to remove
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
;
;       For Find First File,
;
;       Entry   AH = LFN_FUNC
;               AL = 04Eh
;               CX = wildcard search attributes
;               DS:DX = file pathname
;               ES:DI = pointer to search result buffer
;                       (the format of the buffer will be consistent
;                        with the WIN32_FIND_DATA structure)
;       Exit    If successful, carry flag is clear
;                   the search result buffer is filled in
;                   AX = search handle for find next
;               else, carry flag is set
;                   AX = error code
;
;       For Find Next File,
;
;       Entry   AH = LFN_FUNC
;               AL = 04Fh
;               BX = search handle from previous find
;               ES:DI = pointer to search result buffer
;       Exit    If successful, carry flag is clear
;                   the search result buffer is filled in
;               else, carry flag is set
;                   AX = error code
;
;       For Find Close,
;
;       Entry   AH = LFN_FUNC
;               AL = 0??h
;               BX = search handle from previous find
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
;
;       For Create/Open File,
;
;       Entry   AH = LFN_FUNC
;               AL = 06Ch
;               BX = desired access and sharing mode
;               CX = file attributes for create or truncate
;               DX = action to take if file exists or not
;               DS:SI = file pathname
;       Exit    If successful, carry flag is clear
```

```
;                   AX = file handle
;                   CX = returns the action taken
;               else, carry flag is set
;                   AX = error code
;
;       For Rename File,
;
;       Entry   AH = LFN_FUNC
;               AL = 056h
;               DS:DX = original file pathname
;               ES:DI = new filename to rename to
;       Exit    If successful, carry flag is clear
;               else, carry flag is set
;                   AX = error code
;
```

7. Issues

File types will be defined as the last extension in the file name. How these are presented to the user will be determined by the shell.

The "HACK". Since the manipulation of a new LFN file by an old application will often end up deleting the LFN, we may need to consider implementing the following HACK. This HACK is targeted specifically at Word-processors/Editors:

> Many existing Word Processors implement file editing as follows: Make a copy of the file under a TEMP name. Edit the TEMP name. When USER says SAVE, delete the original file, and rename the temp file to the file name that was just deleted. It would be nice if this standard Word Processor/Editor scenario didn't LOOSE the LFN. We will implement this as follows. When a compatibility 8.3 DELETE or RENAME call is made on a file that has an LFN, the DELETED or RENAMED LFN and 8.3 info is cached in a per task cache. If the next INT 21h call that is made is a compatibility 8.3 rename of a file to the same 8.3 name as is stored in the cache, the cached LFN info is added to the just renamed file and the cache is invalidated. If any other INT 21h call is made (ie. not 8.3 rename) the cache is invalidated.

We will need to look carefully at the implementations in all of the major Windows and MS-DOS word processors of this to make sure there is no intervening INT 21h call that is made between the DELETE and the RENAME. We will gain HIGH user benefit if we can get this hack to work for most of the Word Processors (WIN and PC WORD, WORDPERFECT, AMI-PRO).