# SFS

John Hendrikx

**COLLABORATORS**

| | *TITLE* :<br><br>SFS | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | John Hendrikx | July 31, 2024 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# SFS

## 1.1 Smart Filesystem documentation

Smart Filesystem
Version 1.84 BETA
Release 8

http://www.xs4all.nl/~hjohn/SFS



Release date 22 February 2001

FREEWARE

User Manual



## 1.2 Smart Filesystem: Introduction

Smart Filesystem is a new filesystem for your Amiga. A filesystem's main purpose is to store files on a disk in such a way that they can be located and used quickly. How this is done is up to the

filesystem. The way information is stored on your disk has a large impact on speed and space efficiency, and so these factors can vary a lot from filesystem to filesystem.

Smart Filesystem was created to provide you with an alternative to the Fast Fileystem. Smart Filesystem makes more efficient use of your disk space, has better performance in most areas and will allow for new features never seen before in an Amiga filesystem.

COPYRIGHT NOTICE



DISCLAIMER

This version of Smart Filesystem is an early evaluation release, which means it may or may not work on your system. The authors are not responsible for any loss of data or damages to software or hardware that may result directly or indirectly from the use of this software. The author reserves the right to make changes to the software or documentation without notice.

DISTRIBUTION

This version of Smart Filesystem is freeware.

None of the files of the Smart Filesystem package may be modified or left out without permission of the authors. Crunching or archiving is allowed only if none of the files get modified by it.

This version of Smart Filesystem may be distributed freely under the condition that no profit is gained from its distribution.

Permission is granted to distribute this package by Bulletin Board system or network sites, under the condition that no fee is charged on downloading it.

## 1.3 FastView: Features

- Fast reading of directories.
- Fast seeking, even in extremely large files.
- Blocksizes of 512 bytes up to 32768 bytes (32 kB) are supported.
- Supports large partitions. The limit is about 2000 GB, but it can be more depending on the blocksize.
- Support for partitions larger than 4 GB or located (partially) beyond the 4 GB barrier on your drive. There is support for the

New Style Devices which support 64 bit access, the 64-bit trackdisk commands and SCSI direct.

- The length of file and directory names can be 100 characters.
- The size of a file is limited to slightly less than 2 GB.
- Modifying data on your disk is very safe. Even if your system is resetted, crashes or suffers from powerloss your disk will not be corrupted and will not require long validation procedures before you will be able to use it again. In the worst case you will only lose the last few modifications made to the disk. See Safe writing for detailed information on how this works.
- There is a built-in configurable read-ahead cache system which tries to speed up small disk accesses. This cache has as a primary purpose to speed up directory reading but also works very well to speed up files read by applications which use small buffers.
- Disk space is used very efficiently. See the Space efficiency section for a comparison between a few filesystems.
- Supports Notification and Examine All.
- Supports Soft links (hard links are not supported for now).
- Using the SFSformat command you can format your SFS partition with case sensitive or case insensitive file and directory names. Default is case insensitive (like FFS).
- There is a special directory called '.recycled' which contains the last few files which were deleted. See Recycled.

## 1.4 Smart Filesystem: System requirements

Smart Filesystem will only run on Amiga systems equipped with atleast a 68020 processor and Kickstart 2.04 or higher. About 100 kB of memory is the minimum amount of memory Smart Filesystem requires to run.

## 1.5 Smart Filesystem: About the authors

The filesystem is being written by me, John Hendrikx, in SAS/C. I get a lot of help from Marcel Offermans who helps me test the filesystem and provides a lot of valuable input.

The development of the filesystem started in 1993. At that time I wrote a filesystem in assembler without having spend much time on the design. This filesystem never got completely finished although it did function reasonably well at the time. Because of an ever growing assembler source which got more and more complex and because of some

major design flaws this project got halted.

Later on, in 1996, I started writing a filesystem in C instead. I created a decent foundation for a filesystem, but didn't yet work out the specifics in C. After that the C code was left alone for a while and together with Marcel Offermans we've created a design for the filesystem.

By October 1997 I've started to build our ideas using the foundation created in C earlier. The result is what this package is all about.

We can be contacted at:

hjohn@xs4all.nl (John Hendrikx)

For more information and on-line developer documentation, check out:

http://www.xs4all.nl/~hjohn/SFS

## 1.6 Smart Filesystem: Future

Here is a list of things we are planning to add to Smart Filesystem or are under development:

- Multiuser support (muFS) using multiuser.library.
- Built-in background file and free space defragmenter. Already the filesystem is set up in such a way to allow for easy implementation of this feature without having to do extensive scanning of the disk before the defragmenter can begin. This means defragmenting can be done in the background and can be interrupted at any time (even by a reset, crash or power failure) without loss of data.
  [This feature is very near to completion now]
- Mirroring of important filesystem administration blocks to make the filesystem more robust.
- A tool to convert a FFS partition to a SFS partition on the fly.

Suggestions for other features are welcomed. We also welcome any developers wanting to help build support tools for SFS. Especially the FFS to SFS conversion tool is something which we could use some help with.

Below is a list of things we will only implement if there is enough demand for them or if some other developers are willing to help us with them:

- Mirroring of complete partitions for absolute safety.
- Support for striping and special parity stripes.
- Support for hard links (soft links are already supported).
- The ability to extend a partition without having to copy all your

data and format the partition.

- A Partition Magic like tool for the Amiga.

- New DOS packets, or some other new interface to talk with the filesystem. There are lots of ways to exploit a filesystem better than is possible at the moment. New packets for example could be the key to this. Features which would be possible then are paths longer than 255 characters, live directories (directories updated in real time), enforcing recordlocking and many other things.

## 1.7 Smart Filesystem: Space efficiency

Below I've made a comparison of how efficient a few filesystems use their space (based on my knowledge of these filesystems). First however I'll give a short description of each of these filesystems:

FFS (Fast FileSystem)

The standard filesystem for the Amiga. This filesystem stores information for each file and directory separately in one block, regardless of blocksize. It uses lists of blockpointers stored in special blocks to keep track of what blocks belong to a file. FFS uses a bitmap to keep track of free space. Optionally it can cache files and directories together in special blocks (DirCache) for faster directory access.

AFS (AmiFileSafe)

A 3rd party filesystem for the Amiga. Multiple files and directories can be stored in a single block resulting in a more efficient usage of space. AFS uses lists of blockpointers and lengths to keep track of what blocks belong to a file. AFS uses a bitmap to keep track of free space. AFS reserves a fixed amount of diskspace for administration blocks (about 5%). Please note that AFS is no longer available.

FAT16 (FAT = File Allocation Table)

The filesystem used on MS-DOS and Windows based computers. It can store multiple file and directory names in a single block. To keep track of blocks used by files it uses the FAT. The FAT has 1 16-bit entry for every block the disk consists of. This results in a 65536 block limit for FAT16 forcing larger blocksizes for larger partitions. The FAT also serves as a means to keep track of unused space (there is no bitmap).

FAT32

As FAT16, except that the FAT has 32-bit entries instead to allow for partitions with more than 65536 blocks.

SFS (Smart Filesystem)

It is somewhat similair to AFS, but uses slightly different structures and allocates administration space dynamically.

Comparison:

The table gives you an estimate on the amount of overhead in MB for each of the filesystems described above, when they need to store 30000 files varying in size from 4 to 1000 kB. This includes the overhead for the bitmap or FAT and overhead for administration blocks and rounding the size of files up to the nearest multiple of the blocksize. There are about 3 times more small files than large files. The size of the partition used is almost 2 GB.

Keep in mind that these are calculated figures, and that they represent an average case (especially AFS and this filesystem have a bad worse case scenario, which however is extremely unlikely and which can be avoided completely using a file defragmenter). Also keep in mind that a lot of things have been simplified, but these figures should still be accurate to a couple of megabytes.

Overhead in MB

| Blocksize | FFS | AFS | FAT16 | FAT32 | This file-system |
|---|---|---|---|---|---|
| 512 | 47 | 110 | -* | 25 | 11 |
| 1024 | 54 | 117 | - | 24 | 18 |
| 2048 | 91 | - | - | 35 | 33 |
| 8192 | 352 | - | - | 119 | 120 |
| 32768 | 1406** | - | 469 | 470 | 472 |

(*) An '-' indicates the filesystem doesn't support this blocksize at all or just in this case.

(**) The overhead in this case is very extreme. It is unlikely we could fit all 30000 files on the disk as we would run out of space.

If you are interested on how these figures are calculated then read on. The overhead of a filesystem is made up out of a number of components:

1) Each file stored will waste a small portion of space on your disk simply because a file must be stored in a whole number of blocks. Since blocks cannot be used partially for multiple files this means that for every file on the disk about half a block is wasted. The larger the blocksize, the more space is wasted for each file.

| Blocksize | Overhead for 30000 files (in MB) |
|---|---|
| 512 | 7 |
| 1024 | 15 |
| 2048 | 29 |
| 8192 | 117 |
| 32768 | 469 |

2) Some of these filesystems use a bitmap to keep track of what blocks are still free. A bitmap contains 1 bit for every block the disk consists of. The more blocks the larger the bitmap. Increasing the blocksize means less blocks to keep track of and thus means a smaller bitmap.

| Blocksize | Overhead for the bitmap of a 2 GB partition (in kB) |
|---|---|
| 512 | 512 |
| 1024 | 256 |
| 2048 | 128 |
| 8192 | 32 |
| 32768 | 8 |

3) The FAT filesystems do not use a bitmap. The FAT is used to keep track of free space and what space belongs to a file. The FAT is a huge table consisting of 16-bits or 32-bits entries for FAT16 or FAT32 respectively. There is such an entry for every block the partition consists of. Increasing the blocksize means less blocks and thus less entries in the FAT. For FAT16 the table cannot become bigger than 65536 entries, meaning that for a 2 GB partition a blocksize of atleast 32768 bytes is required.

| Blocksize | Overhead for the FAT of a 2 GB partition (in kB) FAT16 | FAT32 |
|---|---|---|
| 512 | - | 16384 |
| 1024 | - | 8192 |
| 2048 | - | 4096 |
| 8192 | - | 1024 |
| 32768 | 128 | 256 |

4) Filesystem specific overhead:

FFS: For FastFileSystem every file stored means that a fileinfoblock is allocated to store its name, comment, protection bits and so on. The larger the blocksize the larger this overhead is. Furthermore, FFS uses special blocks and part of the fileinfoblock to keep track of what data blocks belong to a file. You can compare this to a dynamically allocated FAT. These blocks contain one 32-bit entry for every block the file has in use. The larger the blocksize the less blocks a file will need, and thus less entries are required to keep track of all the blocks.

| Blocksize | Specific overhead for FFS (in MB) |
|---|---|
| 512 | 40 |
| 1024 | 40 |
| 2048 | 62 |
| 8192 | 235 |
| 32768 | 950 |

AFS, FAT16, FAT32 and this filesystem: For these filesystems the rest of the overhead is quite small. All of these filesystems store as much information (name, comment and other information) as possible in each block. The blocksize therefore becomes irrelevant.

FAT16 and FAT32 do not need special blocks to keep track of what blocks a file has in use because the FAT table already contains this information.

AFS and this filesystem do not keep track of what blocks are used by a file by providing a single entry for each block. Instead AFS and this filesystem make use of the fact that usually lots of blocks in sequence belong to a single file. These filesystems simply store the start and end of such a range, which is very compact for the average case. Of course, there is a worst case scenario when storing file allocation information in this way (when a very large file is split up in fragments only 1 block in size) but this scenario is extremely unlikely, and can be avoided by having a file defragmenter.

Finally, AFS uses a fixed amount of a partition (5%) to store all of the information above, except the actual file data itself. This means that it doesn't really matter how much space exactly is used by AFS administration blocks, as a fixed portion of the disk is used for this anyway (I believe there were plans to make this administration area dynamic in size, but as far as I know these were never implemented).

For FAT16 and FAT32 the test-case presented above would mean that about 1-2 MB is wasted on administration blocks. For this filesystem it would mean about 3-4 MB is wasted (with a worst case of more than 30 MB). For AFS it is the same as for this filesystem, except that it doesn't matter as the adminstration blocks are stored in the 5% area anyway.

## 1.8 Smart Filesystem: Reporting problems

Bug reports can be submitted to hjohn@xs4all.nl directly. When submitting bug reports please give us enough information to reproduce the bug if possible. Also do not forget to include a description of your system. Mention atleast the following information:

- Version of Smart Filesystem (you can obtain this by doing a 'version sfs:' where 'sfs:' is the name of the partition you installed Smart Filesystem on)

- Kickstart version, type of processor and memory information. You can use ShowConfig (located in the System drawer) to get this information. Just include the output from ShowConfig in your bug report.

- Type of harddisk controller you are using.

- Please also tell us what type of harddisk you are using the filesystem on when the problem occured (IDE or SCSI), how large the harddisk is and how large the partition is (if possible,

include a mountlist or the partition information as displayed in HDToolBox). You can also use SFScheck which also prints a lot of valuable information.

o Include a detailed description of what error occured, what you were doing at the time and what software you were using. Try to reproduce the problem and let me know if you could reproduce it.

I'll try and confirm bug reports within a day or two.

## 1.9 Smart Filesystem: Known problems

There are a number of things you should know before using this filesystem.

- Do not use disk caching software which delays writes on a SFS disk. PowerCache is known to have this feature (but it can be disabled). SFS relies on data being written in a special order to the disk so it can keep your disk valid it all times. Caching software which delays those writes can therefore interfere with this process.

- Programs using ixemul (GNU C for example) might have problems with this filesystem as well, although this is unlikely.

- Disk changes are implemented, but haven't been fully tested so you might experience problems. The c:DiskChange command might help to avoid some problems by telling the filesystem explicitely that the disk has changed.

- The filesystem doesn't pay attention to write-protection (the filesystem will get confused eventually if you've write protected the disk and start writing data to it).

- Not all space gets freed again if you delete all files from a disk. This is caused by the fact that the filesystem allocates parts of the disk to store its administration blocks on demand. These areas aren't freed again (but they are reused if needed!). This will be fixed eventually, but is no more than a minor inconvience at the moment.

- The filesystem puts up requesters during booting to inform you that last changes to the disk before the last reset weren't completed yet. This means booting may be interrupted and you'll have to confirm the requesters first.

- Although the filesystem supports blocksizes upto 32 kB, it is not recommended to use such large blocksizes. SFS performs very well with small blocksizes and gains very little or even loses speed with larger blocksizes. I'd recommend not using blocksizes larger than 2 kB.

- The structure of future versions of this filesystem WILL change without being backwards compatible as long as the filesytem is in BETA stage. This means you will need to reformat any SFS partitions you have before being able to use a new version. Check

the history to see whether or not you need to reformat your disk.

Don't forget this filesystem is BETA - this means it might crash your machine and damage the files you stored with it. Use it at your own risk and always keep backups of your important data (but that goes without saying anyway).

Check out the Installation section as well for more information which could help to solve problems.

## 1.10 Smart Filesystem: History

30 October 1999, changes for version 1.84:

- When an error occurs during a read or write to the disk, SFS will now automatically retry the operation a few times before reporting an error.

17 October 1999, changes for version 1.83:

SmartFilesystem:

- Fixed Enforcer hit, which could crash the machine.

- Defragmenter can now defragment very fragmented files a bit faster because it can move multiple fragments.

SFSdefragmentGUI:

- Updated SFSdefragmentGUI to support new features of the SFS Defragmenter.

- Added a DEBUG switch to make SFSdefragmentGUI print out what it is doing.

7 October 1999, changes for version 1.81:

- Reduced CPU usage of Defragmenter considerably. Defragmentation should be a lot faster on 020/030 machines now.

- Fixed a bug which was introduced in 1.80. When defragmentation should be completed, SFS would keep on moving some more data.

3 October 1999, changes for version 1.80:

SmartFilesystem:

- Changed Defragmenter to scan for fragments from the end of the disk, instead of from the current point of defragmentation.

- Fixed bug which popped a requester saying something like 'couldn't

mark 128 blocks from block X because some of them were already marked'. This only happened during defragmenting, and it could have destroyed 128 blocks of your disk.

- Added a way to query the version of SFS easily.

SFSdefragmentGUI:

- Now checks SFS version, and refuses to work with older versions.

- Fixed a bug which could crash the machine.

SFSformat:

- Fixed a bug which could crash the machine.

SetCache:

- Fixed a bug which could crash the machine.

19 September 1999, changes for version 1.78:

- Optimized a dozen different routines to improve small file writing speed. CPU usage is also lower in those circumstances now.

- Copyback cache has been improved. The cache doesn't need to read a line anymore before it can do a write to that line (no read-on-write). The read will be delayed until it is needed, or it won't even be performed at all. Previously SFS always did a read-on-write (similair to the 040 copyback cache).

- In 1.58 a bug was introduced which could cause a requester to appear where SFS claims it wants to mark 32 blocks, but some of those were already in use. The requester usually appears when a disk is quite full and quite fragmented. The bug most likely didn't cause any damage (SFScheck reports everything ok after the requester appears), but it was annoying nonetheless. This is now fixed.

- The Defragmenter is now finally debugged far enough to allow it to be BETA tested. The speed problems and the problems which could arise when accessing the disk while it was being defragmented should now be eliminated. More on the defragmenter below.

- SFS now will try to prevent fragmentation when multiple files are written in small bits to a SFS disk at the same time. Previously worst-case fragmentation could occur, where each of 2 files occupied one block after the other. Now SFS tries to make each part of the file atleast 10 blocks large.

- SFS now estimates the size of the Adminstration space it needs

and will write file-data after this reserved size to the disk. The reserved area is not fixed, and when the disk gets full it will simply be used normally. The effect of this is that Administration blocks will get grouped at the start of the disk (as long as there is space there). Previously these Administration areas would be scattered all over the disk.

- A new packet, ACTION_SFS_SET, has been added. See packets.h for details. It allows you to set some parameters of SFS (until reboot for now). No tool is provided yet to set these parameters.

- SFS no longer crashes when used under Kickstart 1.3. It now simply exits without mounting.

- A work-around for a (possible) bug in ixemul.library (47.3) has been added. The bug has to do with Soft Links. This bug only shows up under SFS -- FFS is not affected.

- Changed directory scanning order slightly. This should reduce problems even further with tools which modify entries in a directory while scanning.

- A new version of SFScheck has been added. It sports new parameters LOCK and LINES/READAHEADSIZE

25 April 1999, changes for version 1.62:

- Changed the way SFS handles adding/removing Volumes from the DosList completely. It now tries to add/remove entries from the DosList asynchronously, wherever possible. This could fix all kinds of problems with specific controllers and should also fix the 'not a dos disk' requesters popping up just after SFS was mounted.

10 April 1999, changes for version 1.61:

- The startup-message which the filesystem gets from Dos is now returned *after* SFS has checked the inserted disk.

- Transactions are now stored starting from the place where the disk was last modified, instead of always starting from the beginning. This should speed up flushing of a SFS drive. Bitmap functions have been altered accordingly.

- When deleting a transaction SFS did not take the new OI_DELETE type operations into account -- these are now correctly ignored and removed.

28 March 1999, changes for version 1.58:

- There is an SFSquery tool which displays a bit of information about a SFS drive. Try it.

- SFS will now refuse to mount a partition if it is (partially) located after 4 GB on the drive and there was no 64-bit support available (NSD(64), TD64 or SCSI direct). This helps to prevent damage to your data.

- The read-ahead cache should now be slightly faster on 68040 and 68060 systems by aligning the buffers to 16-byte boundaries.

- Made some alterations to the structure of some SFS blocks to reduce CPU usage and to make the defragmenter easier to add in a new version.

- The default name of the directory where the deleted files are stored is now '.recycled'. It is now referred to as the Recycled directory in all the documentation. Renaming the Recycled directory is of course still possible.

- BUGFIX: Deleting Soft-links now works correctly.

- SFS should now work under Kick 2.04.

- BUGFIX: Truncating files using SetFileSize() could damage the file truncated under some conditions.

- In some cases, calling SetFileSize() twice on an empty file could lead to a damaged file and loss of free space -- fixed.

- BUGFIX: Fixed a small problem with disk changes and DMS.

- There is no seperate SCSI direct version of SFS anymore. SCSI direct will be used automatically when no NSD 64-bit or TD64 support is found (SCSI direct users: please let me know if this works correctly for you -- use the SFSquery tool to find out if SCSI direct is being used).

- BUGFIX: Moving a file from the Recycled directory to the same disk would not lower the file counter which keeps track of how many files are in the Recycled directory. This can lead to files being removed from the Recycled directory too early.

- The contents of the Recycled directory are not considered anymore when calculating the free space.

- BUGFIX: Overwriting empty directories by a file with the same name is no longer allowed.

- SFSformat now has an option to keep the Recycled directory visible if you wish.

- Format will now no longer accept names with colons (':') or slashes ('/') in them.

- Updated SFScheck and SetCache. The source of SFScheck is now also included in the archive.

- Now uses TD_GETGEOMETRY to detect the size of the disk. This means SFS should now work a bit better with DD & HD floppies, MO drives

and any other drives which have disks of different sizes.

- BUGFIX: Renaming a directory into one of its children now generates an error (Object is in use).

- BUGFIX: There was a slight possibility that SFS reported that a valid disk was inserted, but did not yet tell Dos the name of the disk.

- Implemented ACTION_FLUSH.

- Like FFS, SFS now doesn't allow drives to be inhibited which still have pending changes. Also slightly altered the way SFS handles disk insertion and removal.

- BUGFIX: Notification did not correctly check the last character of the notification path which could lead to multiple notifications being sent if there were multiple notifications which only differed by their last character.

Note: This version is NOT compatible with the previous releases of SFS. You'll need to reformat existing SFS partitions if you want to use the new features. It is possible to use the new SFS and the older version at the same time. See Using two versions of SFS.

3 November 1998, changes for version 1.10:

- Set version to 1.0 to be compatible with ReOrg (ReOrg hangs the machine when there is a SFS filesystem which has a 0.x version). SFS is still not finished however, so be careful!

- BUGFIX: Fixed a very rare problem with case-sensitive names where the filesystem could accidently use an object with the same name but with a different case.

- BUGFIX: SFS now allows programs to create files in exclusively locked directories (like FFS does). ACTION_CHANGE_MODE now also works correctly.

- BUGFIX: An implicit delete (when overwriting a file for example) can now no longer cause a notification message to be sent. This should fix problems with IPrefs when ENV: was assigned to a SFS partition.

- BUGFIX: There was a slight chance SFS could accidently return a soft-link if one was encountered while looking for another object.

- NEW: Easy undeleting of files. When formatting a new SFS disk, a special directory (".recycled") will be created. This directory will contain the last 25 files you've recently deleted (if there's enough free disk space). See Recycled for more information.

- Objects with the special Hidden bit set will now no longer appear in directory listings ('.recycled' has Hidden bit set). The Hidden

bit is not the same as the 'h' bit and cannot be changed for now.

- The number of free blocks is now stored in the root block. For backwards compatibility however the number of free blocks is still being calculated by reading the bitmap at mounting time as well. This will be removed in a future version. For old SFS partitions you'll get a requester the first time you use this new version of SFS, which informs you that this number isn't set correctly yet -- this is normal.

- BUGFIX: The Copyback system still had some inconsistencies which could cause corrupt files under specific circumstances. This system has been thoroughly checked and a number of problems where fixed.

- BUGFIX: The Archive bit of some random file sometimes got cleared when deleting a file in the same directory. Instead the Archive bit of the directory itself should have been cleared, but it never was.

- BUGFIX: When the fileptr wasn't in the last fragment of a file and it was extended with SetFileSize() the file contents would be destroyed.

- Sped up reading of small amounts of data (this improves the speed of buffered I/O).

- NEW: Owner UserID and GroupID can now be set (ACTION_SET_OWNER).

- Added lots of sanity checks and thoroughly checked a very large portion of all SFS code and fixed dozens of potential problems.

Note: This version is compatible with the previous releases of SFS starting from version 0.67. There is no need to reformat existing SFS partitions. You won't be able to use the new Recycled feature without reformatting however.

4 August 1998, changes for version 0.85:

- Requesters which ask for a SFS disk to be inserted will now be retried automatically when a disk is actually inserted.

- SFS now guarantees data is commited atleast every 20 seconds, even if the disk is being accessed continuously.

- BUGFIX: The filesystem no longer gets confused when it thinks a disk is inserted twice. This fixes problems with omniscsi.device.

- BUGFIX: Renaming a file to an empty name with the workbench now returns 'object name invalid' instead of 'object not found'.

- BUGFIX: Creating a dir with the same name as a file no longer deletes the file.

- BUGFIX: 100 character limit is now enforced, instead of allowing

you to create names of any length.

- Copyback mode added to internal caching system. This helps a lot to speed up small write operations. I also tweaked the cache system a bit for performance.

- BUGFIX: Creating files larger than 32 MB using SetFileSize() now works correctly. This fixes a problem with creating large file disks using Shapeshifter.

- BUGFIX: Write protection status is now checked each time a disk is inserted, not just at startup.

- BUGFIX: Reading directories has been made a bit more robust which will fix problems with some programs.

- Error messages returned by SFS when there was an error accessing the device are improved.

- Included SFScheck in the archive -- this is a program to check the structure of your SFS disk. It will report any errors it finds but won't make any modifications. Read its docs and use it reguraly if you value your data!

Note: This version is compatible with the previous releases of SFS starting from version 0.67. There is no need to reformat existing SFS partitions.

14 June 1998, changes for version 0.71:

- BUGFIX: It was possible to open a directory as a file using MODE_READWRITE.

- NEW: Soft links are now supported. You need to specify a full path with Makelink when creating soft links. Hard links are not supported.

- NEW: A new program, sfsformat, is now included in the archive. Using this format command you can quick format a SFS disk and specify if you want case sensitive file and directory names.

Note: This version is compatible with the 0.67 and the 0.68 versions. There is no need to reformat existing SFS partitions.

12 June 1998, changes for version 0.68:

- BUGFIX: When using a Mask value which didn't end with a 'F' (like 0x7FFFFFFE for example) then SFS would sometimes damage the data it writes to a file. The problem occured for example when unpacking a LZX archive to a SFS disk.

Note: This version is compatible with the 0.67 version. There is no need to reformat existing SFS partitions.

```
  1 June 1998, changes for version 0.67:

  - First BETA release.
```

## 1.11 Smart Filesystem: Acknowledgments

```
Thanks to Ralph Schmidt and Børge Nøst for helping to debug
some very early versions of the filesystem.  Also thanks to
Sander ter Steege, Eric Sauvageau, Ramiro Garcia and Petter
Nilsen for testing SFS on their machines.

Also thanks to the people who send us bug reports by email
and helped us solve them!
```

## 1.12 Smart Filesystem: Installation

```
    Trying SFS on a ZIP drive
    Installing SFS on a harddisk
    Upgrading from 1.13 or older to a newer version
```

## 1.13 Smart Filesystem: Trying SFS on a ZIP drive

```
If you have a ZIP drive you can quickly try SFS using the "SZ0" icon.
You'll have to specify your SCSI device (ie, scsi.device,
cybscsi.device, etc..) and Unit (5 or 6) with the tooltypes in the
icon first however.
```

## 1.14 Smart Filesystem: Installing SFS on a harddisk

```
Smart Filesystem can be installed in the RDB (Rigid Disk Block) and
automounted.  Standard mountlists are also supported.
Just copy the 'SmartFilesystem' file from the L directory to your L:
directory.  Now create a mountlist, or follow the instructions below.

    Making SFS available for use
    Creating a SFS partition

Below are some links to some more information you may need
when installing SFS.

    Upgrading SFS
    How the Mask field works
    How the MaxTransfer field works
    Notes for drives larger than 4 GB
    What is SCSI direct?
```

## 1.15 Smart Filesystem: Making SFS available for use

To be able to use Smart Filesystem on your harddisk you will first need to put a copy of the filesystem in the Rigid Disk Block (RDB). AmigaOS will then be able to start the filesystem from your harddisk during booting so you can boot from a partition using Smart Filesystem.

To install the Smart Filesystem in the RDB you'll need a Harddisk installation tool like HDToolBox or SCSIConfig.

1) Copy "SmartFilesystem" to your L: directory (this is just so you can locate it easier later on).

2) Start your Harddisk installation tool, select the Harddisk you want to store the Smart Filesystem and go to the screen which allows you to add a filesystem to the disk.

3) On this screen you should be able to select which filesystem to add, select l:SmartFilesystem. You now should be able to set the DosType (or Identifier) field to 0x53465300 ("SFS\0"). You may need to change the filesystem type to 'Custom' first before you can set this.

4) When you're done adding the filesystem, save the changes to disk and exit the program.

If everything went okay you're now done installing Smart Filesystem in the RDB, and it will now be available for use. Go to the Creating a SFS partition section to create a partition which uses Smart Filesystem.

## 1.16 Smart Filesystem: Creating a SFS partition

I'll assume here that you've already installed Smart Filesystem in the RDB by following the instructions in the Making SFS available for use section.

Using your Harddisk installation tool (for example HDToolBox or SCSIConfig) chose the harddisk you previously installed Smart Filesystem on and chose to partition the drive.

Select or create a suitable partition. If you selected an existing partition all data in that partition will be lost when changing it into SFS, so don't forget to make a backup.

You'll need to select which filesystem you want to use on your partition. In HDToolBox this is usually found under Advanced Options where you can use the Change button to go to a screen where you can chose the filesystem. It's also possible there is a cycle gadget where you can select the filesystem.

When selecting the filesystem look for "SFS\0" or 0x53465300 (or something similair) to select the Smart Filesystem for your partition.

Most other fields probably don't need to be changed from their default values. However you may need to change the Mask or Maxtransfer values. Maxtransfer for example should be set to 0x1FFFE for most IDE controllers. If you are in doubt as to what value these 2 fields should have then just copy the values from one of your other partitions from the same drive.

Other fields you might want to set are the name of the partition and maybe the amount of buffers (I'd recommend atleast 100 buffers, more if you have plenty of memory. You can find out the amount of memory required by multiplying the number of buffers with the blocksize).

When you're satisfied with the settings save the changes to the drive and exit the program.

If everything went okay then Smart Filesystem should be available for use on the partition you selected after you've rebooted. All you need to do now is to format the new partition (a quick format will do).

If anything goes wrong and you're computer doesn't start up properly you can use the boot menu to disable your new partition.

## 1.17 Smart Filesystem: Upgrading SFS

If you are upgrading from a previous version of the Smart Filesystem and you installed it in the Rigid Disk Block (RDB) using HDToolBox then copying the new version to L: won't be enough to upgrade the current version. Instead you'll need to use HDToolBox to remove the old version from the RDB and add the new one (in HDToolBox there is an Upgrade Filesystem option for this purpose).

For instructions on adding a filesystem to the RDB see the Making SFS available for use section.

## 1.18 Smart Filesystem: Example mountlist

You'll need to set a lot of fields yourself to get it to work. Below first is a list of fields you should include unchanged in your mountlist:

```
Filesystem      = L:SmartFilesystem
Flags           = 0
Reserved        = 2
Interleave      = 0
Globvec         = -1
Dostype         = 0x53465300
```

Now follows an example mountlist:

```
SFS:
Device          = <device name, 'scsi.device', 'cybscsi.device' etc...>
```

```
Unit            = <unit number>
Filesystem      = L:SmartFilesystem
Flags           = 0
Reserved        = 2
Interleave      = 0
Globvec         = -1
Dostype         = 0x53465300
BlockSize       = 512
Surfaces        = <fill in your partition in these fields>
BlocksPerTrack  =
Lowcyl          =
Highcyl         =
Buffers         = 200
BufMemType      = 0
Mask            = 0x7FFFFFFF
Maxtransfer     = 0x100000
Mount           = 1
#
```

Don’t forget that the Mask and MaxTransfer value depend on your controller. If in doubt please check what you are using for your other partitions and use those values.

If you have for example a SCSI ZIP drive connected to a SCSI device named ’cybscsi.device’ and its unit is set to 5 then you can use this mountlist:

```
SFSZIP:
Device          = cybscsi.device
Filesystem      = L:SmartFilesystem
Unit            = 5
Flags           = 0
Reserved        = 2
Interleave      = 0
Globvec         = -1
Dostype         = 0x53465300
BlockSize       = 512
Surfaces        = 2
BlocksPerTrack  = 34
Lowcyl          = 2
Highcyl         = 2890
Buffers         = 128
BufMemType      = 0
Mask            = 0x7FFFFFFF
Maxtransfer     = 0x100000
Mount           = 1
#
```

## 1.19 The MaxTransfer field

The MaxTransfer field can be used to tell a filesystem that the device which comes with your (harddisk) controller can’t handle more than a specific amount of data in a single access. This problem usually occurs with IDE drives, which usually have a limit of 64 or 128 kB which can be transfered at once.

When a device has been properly written it should be able to cope with any amount of data being transfered. These devices can have a MaxTransfer value of 0x7FFFFFFF. Only badly written or very old devices need to set a smaller value in MaxTransfer -- in other words, the MaxTransfer value is a compatibility kludge to fix broken devices.

In any case, if you have a SCSI drive, then a MaxTransfer value of 0x7FFFFFFF should be just fine. For IDE drives, you probably need to set it to 0x1FFFE or to 0xFFFE. Those values represent 128 kB minus 2 bytes and 64 kB minus 2 bytes respectively.

An incorrect MaxTransfer value can usually be detected by copying a few large files (more than 200 kB) to such a partition. If the large files are damaged while smaller files are undamaged then this is usually an indication that the MaxTransfer value is too large.

Remember, the MaxTransfer value needs to be set for each partition. Just changing one MaxTransfer value will only affect a single partition, not the entire drive.

## 1.20 The Mask field

The Mask field can be used to tell a filesystem that the device which comes with your (harddisk) controller cannot directly access its data in all regions of memory available on your system.

When a device has been properly written it should be able to cope with data located anywhere in memory. For those devices the Mask should be set to 0xFFFFFFFF. Only badly written or very old devices need a different Mask -- in other words, the Mask value is a compatibility kludge to fix broken devices.

For example, some devices can't access data starting at an uneven address in memory. Some even can only access data when it starts at an address which can be divided by four. In the first case you would set the Mask field to end in 'FFFE', and in the second case to 'FFFC'. If your controller can handle addresses without alignment restrictions then you can set it to 'FFFF' (which is of course the preferred value).

There are also devices which can only access memory in the 24-bit memory area (everything below the 16 MB boundary). Usually these are Zorro-II controllers which cannot directly access memory located on, for example, an accelerator card. For these devices you set the mask to 0x00FFFFFF, indicating that the device can only access data in the 24-bit address space.

Devices which can access data located anywhere in memory (a SCSI controller which is embedded on an accelerator card, or a Zorro-III IDE or SCSI controller) should have a mask of 0xFFFFFFFF.

Here is an overview to clarify the Mask setting:

xxxxxxxF - Use a Mask ending with a 'F' if you're device is written

correctly and can handle transfers to and from memory with any alignment.

xxxxxxxE - Use this if you're device can only handle 16-bit or WORD aligned transfers.

xxxxxxxC - Use this if you're device can only handle 32-bit or LONG aligned transfers.

FFFFFFFx - Use this Mask if you're device is written correctly and can work with any memory in the system. The first 'F' may also be '7' since there are no Amiga's which can have more than 2 GB of memory.

00FFFFFx - This Mask restricts transfers to the 24-bit address space, meaning it can only access ChipRAM and FastRAM in the 24-bit area (The 24-bit area is 0x00000000 to 0x00FFFFFF).

001FFFFx - This Mask restricts transfers to ChipRAM only (the old trackdisk.device needed this for example).

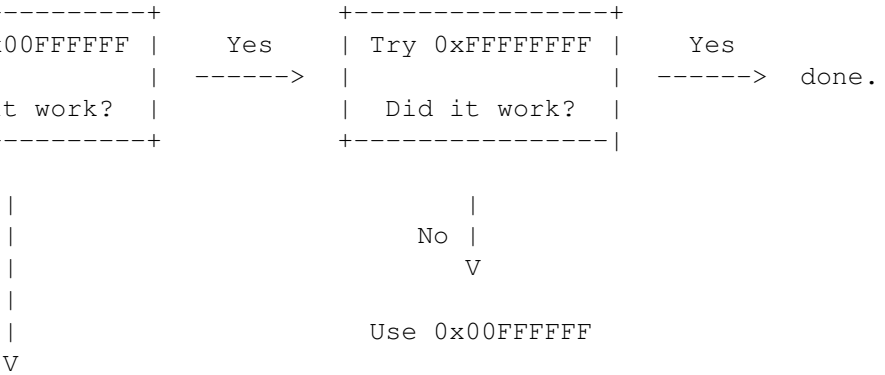
Always use the least restrictive Mask possible. The ideal Mask is 0xFFFFFFFF.

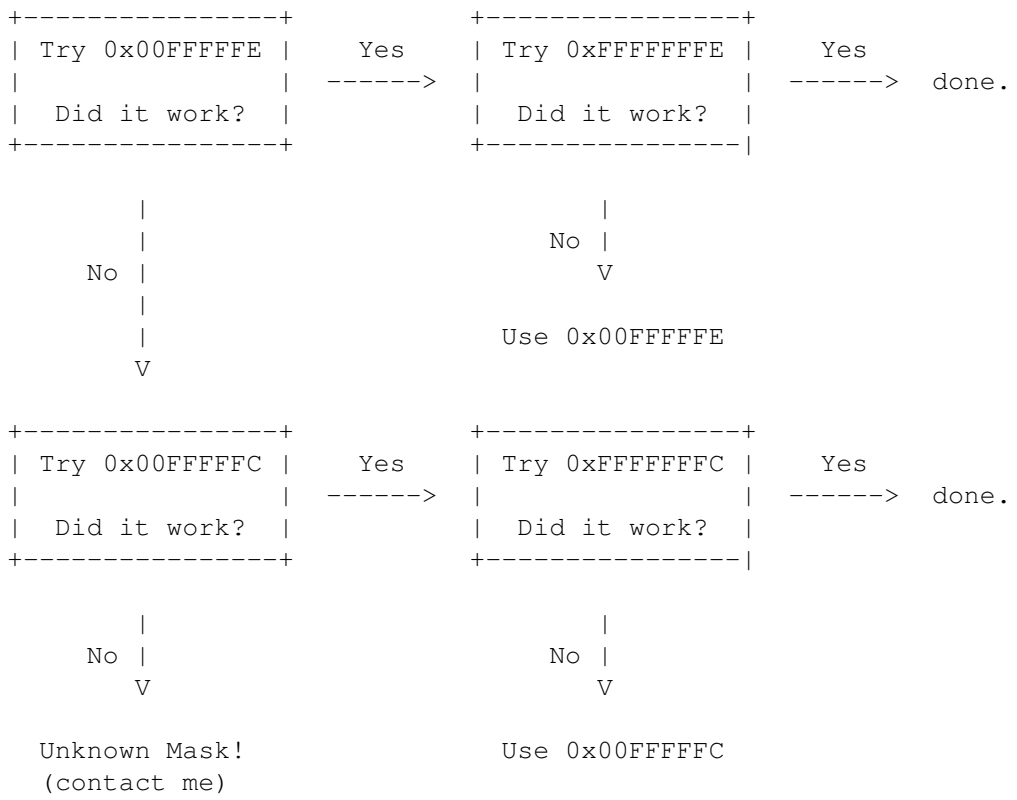
Remember, the Mask value needs to be set for each partition. Just changing one Mask value will only affect a single partition, not the entire drive.

## What Mask setting should I use?

If you're in doubt, check your controller's manual and find out what Mask setting they recommend. You can also experiment a bit with different Mask values, but you got to be careful there. Preferably experiment with a dummy partition which doesn't contain any important data.

If you intend to experiment, follow the scheme below to find out the best Mask value for you. To determine if a specific Mask value works, you'll need to read and write some data to the disk. Copying files is not enough. Preferably unpack a few LhA or LZX archives to the disk, and see if they are unpacked undamaged.






## 1.21 Internal workings of Smart Filesystem

Safe writing A detailed explanation of how SFS makes sure that your disk never gets damaged or invalidated by crashes or power failure.

## 1.22 Safe writing

Overview

The filesystem ensures that its structure is never in an invalid state on the disk. This includes things like the bitmap, the directory tree and file information (for example size and protection bits). Data blocks, the space which contains the data stored in a file, are however not kept completely valid at all times for performance reasons.

The filesystem keeps its structure valid by never overwriting blocks directly. This means that even if a crash or power loss occurs that the old structure will still be present on the disk. When rebooting your machine SFS will be able to detect if changes were pending and will either discard them if they weren't completed yet or finish the pending changes.

However, I already mentioned that SFS doesn't do this for data blocks. This means that if a crash occurs it is possible that some of the data which you were writing to a file has been lost or has partially overwritten existing data in that file.

In the worst case this means the following: For example, take a file of 1000 bytes. The last action you did before the crash was to write 2000 bytes from position 500; in other words the first 500 bytes are unmodified and the new file size becomes 2500 bytes.

When a crash occurs immediately after this write action the filesize will still be 1000 bytes, however the bytes from position 500 to 999 will have been overwritten with new data. The reason that the filesize won't have changed yet is because these changes were discarded to keep the structure of the disk valid. The 500 overwritten bytes however were written immediately and can't be recovered.

## Internals

As was said, Smart Filesystem only ensures that its own structures are kept valid. To do this it keeps track of all changes made to this structure. If a filesize needs to be updated in a specific block, then we add this change to a list of changes to be made. This list is kept in memory until the time comes to commit these changes to the disk. The same goes for all other changes made to the filesystem structure. They are all recorded and added to the list in memory.

The caching system in SFS is smart enough to distinguish between original blocks and blocks with the latest changes applied to them. Also when reading new blocks from disk SFS will automatically apply any changes to these blocks (if any) before using them for internal operations.

All changes which belong to the same operation are kept together. Creating a new empty file for example will result in a number of small changes. A fileheader is created, the file is given a node number and the file is linked into a hash chain. Either ALL these changes are added to the changes buffer or NONE at all.

The way the changes are stored in memory is very simple. SFS compares the original and modified version of a block and stores the difference between them using a quick and very simple compression scheme. This keeps memory consumption low and also speeds up writing the changes to disk since they take of far less space using this simple compression technique.

When the time comes to commit the changes to disk, then SFS will first look for a free area on the disk (SFS automatically ensures there is always enough free space for

this). In this free area it writes the buffer of changes in its compressed form. When this buffer was written correctly, a special block is written to a fixed location. This block is called the Transaction Failure block.

The Transaction Failure block points to the compressed changes which were written earlier to free areas of the disk. The mere presence of this block indicates that there are pending changes in compressed form on the disk. Its presence indicates that the last changes in the transaction weren't completed yet, hence its name.

After writing the compressed changes and the Transaction Failure block, SFS will start to make the actual changes to its structure on the disk. It will simply overwrite existing blocks now, replacing them with their updated versions.

If this process is interupted then next time SFS is started it will see the Transaction Failure block. It will load the compressed changes from the free area of the disk and continue to make the changes (changes already made are simply made again). You could compare this to the validating process of FFS, but you'll never notice since this will usually take only a fraction of a second to complete.

If the process was interupted before the Transaction Failure block was written, then no changes will have been made yet and SFS will simply use the old structure (this in effect discards the last changes made to the disk).

If however everything went smoothly and the system didn't crash during this procedure the Transaction Failure block will be removed again, which indicates the disk is in a valid state. The whole process of updating the disk in this way usually takes less than a second.

## Assumptions

Smart Filesystem makes a few assumptions to be able to guarantee that the system of keeping your disk valid at all times works:

- Writing a single block is atomic. This means either the block was physically written completely to disk, or not at all. Checksums are used here for extra safety should this operation not be atomic (I haven't been able to confirm or deny this yet for hard drives -- such information seems to be hard to find).

- Write Caching is disabled -- this means that everything written to disk (particularly the changes buffer) was indeed immediately written physically before any other blocks are written. There is a very delicate order here in which things need to get written to be able to

guarantee it works. See below.

- Device drivers which have internal buffers must respect the CMD_UPDATE command which flushes the internal buffers to disk immediately. SFS will use CMD_UPDATE before and after any critical operations.

Order in which things must be written:
---

1. Writing all changes to empty areas on the disk.

2. Writing the Transaction Failure block which indicates there is a valid but unfinished set of compressed changes on the disk. This block points to the blocks stored under step 1.

3. Applying the real modifications to the disk, replacing any blocks which need to be modified.

4. Removing the Transaction Failure block.

Smart Filesystem assumes that ALL blocks written in each of the steps above were physically written before blocks of any of the following steps are physically written to disk. Between the steps SFS will call CMD_UPDATE to flush any buffers the device driver might be using (trackdisk.device does this for example).

Final words

This system is quite safe, but there is the slight possibility that things go wrong anyway if any of the assumptions Smart Filesystem makes isn't met. Backing up your important data is therefore still important, no matter how safe the filesystem. Even if the chance of failure by crash or power loss has in theory been reduced to fractions of a percent, then there is still the possibility of fatal bugs in the filesystem or bad sectors on your disk.

## 1.23 Notes for drives larger than 4 GB

To use a drive which is larger than 4 GB (4096 MB) you'll need to have a filesystem which supports such drives and you'll need a device which can handle these drives.

Getting a filesystem which supports drives larger than 4 GB is easy. Smart Filesystem can handle such drives correctly. There is also a patch or upgrade freely available for FastFilesystem. This will upgrade your version to V43 or V44.

Now you need to make sure that your device also supports drives larger than 4 GB. If your device supports any of the following, then it should be able to handle such drives:

- Your device is a New Style Device (NSD) which supports 64-bit addressing.

- Your device is TD64 compatible.

- Your device supports SCSI direct access. Even devices for IDE harddisks can support this.

Of course you'll need to make sure that your device and the filesystem your using speak the same language. If you've got a device which supports SCSI direct, but doesn't support NSD or TD64 then you won't be able to use this device for drives larger than 4 GB with a filesystem which doesn't support SCSI direct but only supports NSD or TD64.

Therefore I've included two lists so you can see which protocols some Devices and Filesystems support. The list aren't complete. I'll need your help to extend the lists, so if you have got more information send me an e-mail.

| DEVICES | Ver. | NSD(64) | TD64 | SCSIdirect |
|---|---|---|---|---|
| scsi.device (A1200/A4000) | 43.21 | yes | no | yes |
| cybscsi.device (Cyberstorm) | 8.1 | no | yes | yes |
| HardFrame.device (Microbotics) | 1.5 | ? | no | yes |
| statram.device (Ram Disk) | ? | ? | no | no |
| ramdrive.device (Ram Disk) | ? | ? | no | no |
| scsidev.device (GVP Series I) | ? | ? | ? | no |
| hddisk.device (CBM A2090) | ? | ? | ? | no |

| FILESYSTEMS | Ver. | NSD(64) | TD64 | SCSIdirect |
|---|---|---|---|---|
| FastFilesystem | 43.18 | yes | no | ? |
| FastFilesystem | 44.5 | no | yes | yes |
| Smart Filesystem | 0.71+ | yes | yes | yes(*) |

(*) Use the SCSI direct version.

It's a very good idea to add some checks to your startup-sequence to see if your using the correct version of your device and filesystem. If for any reason you're using the wrong versions then you could easily end up destroying your data. That's why I have added these lines to my User-startup:

```
Version DH1: VERSION=44 >NIL:
IF WARN
  ECHO "Warning!  V44 FastFilesystem is not loaded!"
  Version DH1:
ENDIF
```

```
 Version scsi.device VERSION=43 >NIL:
 IF WARN
   ECHO "Warning!  V43 scsi.device is not loaded!"
   Version scsi.device
 ENDIF
```

You'll might need to modify these checks a bit for your own setup.

Finally, don't use tools like ReOrg, DiskSalv, DynamiCache, PowerCache, AmiBackTools, QuarterbackTools and similair tools on partitions which are located after the 4 GB border. They don't support drives larger than 4 GB and will destroy data on other partitions if you use them!

Always be absolutely sure your tool supports drives larger than 4 GB before using them! If you've got a tool and it reports errors or acts strangely when working with one of your partitions after the 4 GB border (for example, DiskSalv doesn't recognize that a disk after the 4 GB border is FFS) then that tool probably doesn't support drives larger than 4 GB.

Also be careful with the standard Format command! Always use the QUICK option for drives larger than 4 GB, otherwise format may format information in the wrong area of your disk!

See What is SCSI direct for more information on SCSI direct and how it could be useful for drives larger than 4 GB.

## 1.24 What is SCSI direct?

In the archive there were 2 versions of Smart Filesystem available, a normal version and a SCSI direct version.

The normal version talks to your device in a way which all devices support (with device I mean for example 'scsi.device', 'omniscsi.device' or 'cybscsi.device' not the drive itself). The normal version will automatically detect New Style Devices (NSD) and TD64 devices which support drives larger than 4 GB. If neither is detected then it will use the standard way of communicating with your device which means you are limited to harddisks of 2 GB or 4 GB in size (depending on what your device supports).

SCSI direct is just another way of communicating with your device. Even some IDE devices support the SCSI direct protocol, which they translate automatically to normal IDE commands. For example, the standard scsi.device which comes with A1200's and A4000's with IDE on board understand SCSI direct.

The advantage of the SCSI direct protocol is that it can work with harddisks larger than 4 GB as well. So if your device isn't NSD or TD64 compatible then you still have the option of using SCSI direct to use disks larger than 4 GB.

SCSI direct doesn't give you a performance boost. It is just as fast as the normal version of SFS. The only difference is in the way SFS communicates with your device, and you'll only need it if you've got a drive larger than 4 GB and your device doesn't support NSD or TD64.

See notes for drives larger than 4 GB for more information on large drives.

## 1.25 The '.recycled' directory

SFS supports a special directory (".recycled") which contains the files you most recently deleted. This directory allows you to quickly and easily recover a file you deleted by accident.

Undeleting a file is a matter of moving or copying the file from the Recycled directory to a different location. Be careful when moving or copying files to the same disk however, since SFS may decide at any time that it needs to make room for new files which could cause the file(s) you're trying recover to get removed.

The contents of the this special directory is maintained by the filesystem. You're not allowed to move files there or create new files in this directory. Files stored in the Recycled directory will automatically be deleted when the disk gets full, or when the directory contains more than 25 files.

You're not allowed to change the contents of files stored in the Recycled directory; this also goes for their name, comment, protection bits and date.

The Recycled directory itself will only get created during formatting. It is not possible to create a '.recycled' directory afterwards (atleast not for now). The Recycled directory can't be deleted, but you are allowed to rename it. It is even possible to move it into a subdirectory.

FLUSHING THE RECYCLED DIRECTORY

You can easily permanently remove files by going to the Recycled directory and deleting any files there. SFS will detect this and will remove the file permanently.

## 1.26 masks

| | Mask | MaxTransfer |
|---|---|---|
| scsi.device (A4000, IDE) | 0xFFFFFFFE | 0x1FFFE |
| cybscsi.device | 0xFFFFFFFF | 0xFFFFFFFF |
| DKB 4091 | 0xFFFFFFFE | 0xFFFFFFFF |

## 1.27 Using two versions of SFS simultaneously

When upgrading to a new SFS version while SFS is in BETA I can't always keep SFS fully compatible to older versions. This means that at some point you'll have to reformat your SFS partitions to be able to use the latest version.

However, it is possible to run two different versions of SFS at the same time -- you simply need to treat this new version of SFS as a completely new filesystem. So just like FFS can't be directly replaced by a SFS filesystem, this new version can't simply replace the old SFS version either.

Version 1.13 or older of SFS is not compatible with versions released after 1.13.

SFS in the RDB

If you've put the old version of SFS in the RDB then you can add the new version there as well. Don't remove the old version until you've converted all of your SFS partitions.

Add the new version, but give it a different DosType. Normally you used 0x53465300 (=SFS/0), but give the new version a different DosType. I'd recommend using 0x53465301 (=SFS/1). Now you can choose between the old SFS version (SFS/0) and the new one (SFS/1) for each of your partitions.

If you changed one of your old SFS partitions to the new version, SFS will put up a requester during booting telling you that the SFS disk is in a format which is not supported anymore. This is normal. After formatting the disk it should be useable again. You can use SFSformat or the normal Format command with the QUICK option to format the disk.

SFS with mountlists

If you mount your SFS partitions from a mountlist then you can also use two versions of SFS at the same time. Just copy the new version of SFS to your L: directory with a different name (for example 'SmartFilesystem2'). Now in your mountlists you can use the new version of SFS by changing the following lines:

```
Filesystem = l:SmartFilesystem2
DosType = 0x53465301
```

Also take a look at the Installation section for more
details.