

From: Laurent.Clevy@meteo.fr
Newsgroups: comp.sys.amiga.programmer, comp.sys.amiga.misc, comp.os.linux.misc
Subject: Amiga floppy disks format (AmigaDos file system - floppies)
Followup-To: poster
Summary: This document describes the AmigaDos File System for floppy disks only.

Physical/Logical formats, OFS/FFS, Directory caching, Links, Checksums.

Archive-name: amiga/amiga_floppy_format
Last-modified: 28. May 1997
Version: 0.9
Copyright: (c) 1997 Laurent Clevy
Maintainer: Laurent Clevy <Laurent.Clevy@meteo.fr>

FAQ : The Amiga floppy disks format
Laurent Clevy

Laurent.Clevy@meteo.fr
25 avenue Aristide Briand
28000 Chartres
France

Disclaimer and copyright

This document is Copyright (C) 1997 by Laurent Clevy, but may be freely distributed, provided the author name and addresses are included and no money is charged for this document.

This document is provided "as is". No warranties are made as to its correctness.

Amiga, AmigaDos are registred trademarks of Gateway 2000.

Introduction

This document purpose is to describe the Amiga floppy disk format. I don't found any document which explains this format in details. Because I wish this machine to be supported a long time, including via emulators, I decided to write this file, and supply C routines as examples.

Corrections (including about my english) are very welcome. Unfortunately, I have no permanent e-mail address currently, the only way to touch me is by postmail.

Index

1. How bytes are physically read/written on disk ?
 - 1.1 What is MFM ?
 - 1.2 What is the physical track format ?
 - 1.3 What is the physical sector format ?
 - 1.4 How to decode MFM data ?
2. What is the Amiga floppy disk geometry ?
3. How is logically organised a Amiga floppy disk ?
 - 3.1 What is a Bootblock ?
 - 3.2 What is a Rootblock ?
 - 3.2.1 How to find the first sector of a entry ?
 - 3.2.2 How to list directory entries ?
 - 3.2.3 How to compute the checksum ?
 - 3.3 How is managed the free/used blocks list ?
 - 3.3.1 How to compute bitmap checksum ?
 - 3.3.2 What is the 'bm_ext' field in Rootblock ?
 - 3.4 How are stored files on a disk ?
 - 3.5 How are stored directories ?
 - 3.6 How are implemented links with AmigaDos ?
 - 3.6.1 Hard links
 - 3.6.2 Soft links
 - 3.7 How is the block associated to directory caching ?
4. What is a blank disk ?
 - 4.1 a Minimal blank disk
 - 4.2 A 'Bootable' disk
 - 4.3 A Directory cache mode disk
5. References
6. C Routines
7. Other Amiga file systems
8. To do

Conventions

* In this document, hexadecimal values are written like in C : for example 0x0c is the decimal value 12.

As the Amiga is a 680x0 based computer, integers that require more than

one byte are stored on disk in 'Motorola order' : the most significant byte comes first, then the less significant bytes in descending order of significance (MSB LSB for two-byte integers, B3 B2 B1 B0 for four-byte integers). A byte is 8 bits long. The left bit of a byte is the 7th, the right bit is the 0th.

A 'word' is a 2 bytes (16 bits) integer, a 'long' a 4 bytes (32 bits) integer. Values are unsigned unless otherwise noted.

* A block pointer is the number of this block on the disk. Disk starts with the #0 block.

* Hashing is a method to access tables : given a number or a string, a hash function gives a index into a table.

* Chained lists are cells oriented data structures. Each cell contains a pointer to the next or previous cell or both, the last pointer is null.

C example :

```
struct lcell {
    char name[10];
        /* contains next cell adress, or NULL if this cell is the last
*/
    struct lcell* next_cell;
};
```

* Names of blocks begin with a capital (Rootblock).
Name of fields are noted between quotes ('field_name').

* All formats are described as tables, one rows per field.
Here is an example with then well known beginning of GIF format :

offset	type	length	name	comments
0	char	3	signature	'GIF'
3	char	3	version	'87a' or '89a'
6	short	1	screen width	
8	short	1	screen height	

1. How bytes are physically read/written on disk ?

=====

Most of PC-like floppy disk controllers (FDC) are not able to read Amiga disks, because Amiga physical floppy disk operations are made by a specific chip called "Paula".

However, i'm supplying this information because it's hard to find out.

If you only want to understand the UAE .adf format, you don't need to read this part.

For classical floppy disk operations, Paula is set with the following parameters :

- MFM encoding
- Precompensation time : 0 nanosec
- Controller clock rate : 2 microseconds per bit cell
- Sync value = 0x4489

The controller is able to put the read/write heads on a cylinder, and is able to read with the lower or upper side head. A track of 0x1900 words is usually read.

1.1 What is MFM ?

Because bits can't be written with magnetic fields directly on disk, an encoding scheme is required. Amiga floppy disks are MFM (Modified Frequency Modulation) encoded.

The MFM encoding scheme is :

user's data bit	MFM coded bits
-----	-----
1	01
0	10 if following a 0 data bit
0	00 if following a 1 data bit

User data longs are splitted in two parts, a part with even bits part first, followed by a part with odd bits.

1.2 What is the physical track format ?

Double density (DD) disks have 11 sectors per track, High density (DD) disks have 22.

So a track consists of 11/22 MFM encoded sectors, plus inter-track-gap. Note that sectors are not written from 0 to 10/21, you must use the 'info' field to recreate orderly track in memory. Each track begins with the first sector, and ends the end of the last sector (11th with DD disks, 22th with HDs).

Each sector starts with 2 synchronization words. The synchro value is 0x4489.

1.3 What is the physical sector format ?

Here it comes :

0/0x0 word 2 MFM value 0xAAAA AAAA

SYNCHRONIZATION

4/0x4 word 1 MFM value 0x4489

6/0x6 word 1 MFM value 0x4489

HEADER

8/0x8 long 1 info (even bits)

12/0xc long 1 info (odd bits)

decoded long is : 0xff TT SS SG

TT = track number (3 means cyl 1, head 1)

SS = sector number (0 -> 10/21)

sectors are not orderly !!!

SG = number of sector before gap (including current one)

Example for cylinder 0, head 1 of a DD disk :

0xff010009

0xff010108

0xff010207

0xff010306

0xff010405

0xff010504

0xff010603

0xff010702

0xff010801

<-- inter-sector-gap here !

0xff01090b (0xb means -1 ?)

0xff010a0a (0xa means -2 ?)

```
16/0x10 long    4      sector label (even)
32/0x20 long    4      sector label (odd)
                    decoded value seems to be always 0

    END OF HEADER

48/0x30 long    1      header checksum (even)
52/0x34 long    1      header checksum (odd)
                    (computed on mfm longs,
                    longs between offsets 8 and 44
                    == 2*(1+4) longs)

56/0x38 long    1      data checksum (even)
60/0x3c long    1      data checksum (odd)
                    (from 64 to 1088 == 2*512 longs)

    DATA

64/0x40 long    512    coded data (even)
576/240 long    512    coded data (odd)
1088/440
    END OF DATA
```

1.4 How to decode MFM data ?

the algorithm :

```
#define MASK 0x55555555 /* 01010101 ... 01010101 */
unsigned long *p1;      /* MFM coded data buffer (size == 2*data_size) */
unsigned long *q;       /* decoded data buffer (size == data_size) */
unsigned long a,b;
unsigned long chksum;
int data_size;         /* size in long, 1 for info, 4 for sector label */

chksum=0L;
do <data_size> times {
    a = *p1;            /* even bits */
    b = *(p1+data_size); /* odd bits */
    chksum^=a;         /* eor */
    chksum^=b;
    *q = ( b & MASK ) | ( ( a & MASK ) << 1 ); /* MFM decoding */
    p1++;
    q++;
}
chksum&=MASK;
```

2. What is the Amiga floppy disk geometry ?

=====

Here follows the disk geometries for DD and HD.

	bytes/sector	sector/cyl	sides/cyl	cyl/disk
DD disks	512	11	2	80
HD disks	512	22	2	80

The relations between sectors, sides and cylinders are for a DD disk :

Block	sector	side	track
0	0	0	0
1	1	0	0
2	2	0	0
...			
10	10	0	0
11	0	1	0
...			
21	10	1	0
22	0	0	1
..			
1759	10	1	79

A DD disk has $11*2*80=1760$ (0 to 1759) blocks, a HD disk has $22*2*80=3520$ blocks.

Of course the file system uses some of them, even for a blank disk. As the next part deals with, at least 4 blocks are used, for 3 logical structures : bootblock (2), rootblock (1) and bitmap block (1).

The length of .ADF files for a DD disk is then $512*11*2*80 = 901120$ bytes.

3. How is logically organised a Amiga floppy disk ?

=====

The logical low level object of a Amiga disk is the 'sector' (or 'block') : 512

consecutive bytes.

Disk information is distributed in the Bootblocks, the Rootblock and Bitmap block(s).

FFS has block structures to provide directory list caching and (hard) links : Directory cache blocks and Link blocks.

Directory tree is stored with a Directory block for each node. Directory entries (files, directories and links) are stored with a table, and are accessed with hashing and chained lists.

Files are stored with a File header block and Data blocks. File extension blocks are also used for files stored with more than 72 Data blocks.

3.1 What is a Bootblock ?

The first object of an Amiga floppy is the Boot block. If the checksum and the DiskType are correct, the system will execute the bootblock code, at boot time, of course :-).

A valid bootblock is written by the AmigaDos command 'install'.

* BootBlock (1024 bytes) sectors 0 and 1

offset	size	number	name	meaning
0/0	char	4	DiskType	'D''O''S' + flags (0->5) flags = set clr 0 FFS OFS 1 INT NOINT 2 DIRC NODIRC
4/4	long	1	Chksum	special checksum
8/8	long	1	Rootblock	==880 DD and HD
12/0x0c	char	1012	Bootblock code	(see 4.2 'Bootable disk' for more information)

The DiskType flag informs of the disk format.

OFS = Old/Original File System, the first one. (AmigaDos 1.2)

FFS = Fast File System (AmigaDos 2.04)

INT = International characters Mode (AmigaDos 3.0).

DIRC = stands for Directory Cache Mode (AmigaDos 3.0). This mode speeds up directory listing, but take some disk space.

There are few differences between the two file systems.

- OFS Datablock stores 488 bytes, FFS stores 512 bytes,
- FFS supports directories caching, links and international mode,
- the FFS is faster than OFS.

The bootblock checksum algorithm follows :

* in 68000 assembler :

```
        lea    bootbuffer,a0
        move.l a0,a1
        clr.l  4(a1)                ;clear the checksum
        move.w #256-1,d1            ;1024/4 times
        moveq  #0,d0
lpchk:  add.l  (a0)+,d0              ;accumulation
        bcc.s  jump                ;if carry set, add 1 to checksum
        add.l  #1,d0
jump:   dbf   d1,lpchk              ;next long word

        not.l  d0
        move.l d0,4(a1)            ;new checksum
```

* in C :

```
#include<limits.h>
#define Short(p) ((p)[0]<<8 | (p)[1])
#define Long(p) (Short(p)<<16 | Short(p+2))

unsigned long newsum,d;
unsigned char buf[1024];          /* contains bootblock */
int i;

memset(buf+4,0,4);               /* clear old checksum */
newsum=0L;
for(i=0; i<256; i++) {
    d=Long(buf+i*4);
    if ( (ULONG_MAX-newsum) < d ) /* overflow */
        newsum++;
    newsum+=d;
}

newsum=~newsum;
```

3.2 What is a Rootblock ?

The Rootblock is at the middle of the media : block number 880 for DD disks, block 1760 for HDs.

The Rootblock contains information about disk : its name, its formatting date, etc ...

It also contains information to access the files/directories/links located at the root (Unix /) directory.

* Rootblock (512 bytes) sector 880 for a DD disk, 1760 for a HD disk

```
-----
```

0/0	long	1	type	block primary type = T_HEADER (value 2)
4/4	long	1	header_key	unused in rootblock (value 0)
	long	1	high_seq	unused (value 0)
12/c	long	1	ht_size	Hash table size in long (value 0x48)
16/10	long	1	first_data	unused (value 0)
20/14	long	1	chksum	sum to check block integrity
24/18	long	72	ht[]	hash table (entry block number)
312/138	long	1	bm_flag	bitmap flag, -1 means VALID
316/13c	long	25	bm_pages[]	bitmap blocks pointers (first at 0)
416/1a0	long	1	bm_ext	first bitmap extension block (Hard disks only)
	...			
432/1b0	char	1	name_len;	disk name length
433/1b1	char	30	diskname[]	disk name
	...			
472/1d8	long	1	days	last access date : days since 1 jan 1978
476/1dc	long	1	mins	minutes past midnight
480/1e0	long	1	ticks	ticks (1/50 sec) past last minute
484/1e4	long	1	c_days	creation date
488/1e8	long	1	c_mins	
492/1ec	long	1	c_ticks	
	long	1	next_hash	unused (value = 0)
	long	1	parent_dir	unused (value = 0)
504/1f8	long	1	extension	FFS: first directory cache block, 0 otherwise
508/1fc	long	1	sec_type	block secondary type = ST_ROOT (value 1)

```
-----
```

3.2.1 How to find the first sector of a directory entry ?

Given the name of a file/directory/link you compute its hash value with this algorithm :

* The hash function :

```
#include<ctype.h>

int HashName(char *name)
{
int hash;
int i,l;

l=hash=strlen(name);
for(i=0; i<l; i++) {
    hash=hash*13;
    hash=hash + toupper(name[i]);
    hash=hash & 0x7ff;
}
hash=hash % 72;
hash=hash + 6;

return(hash);
}
```

The hash value is used to access HashTable ('ht' field in Rootblock/Directory block).

But several names can result a unique HashValue. The first block pointers of them are stored in a chained list which start at HashTable[HashValue-6].

Here follows the method to find the requested block :

1. HashValue = HashName(name)
2. sector = Hashtable[HashValue-6]
3. if (sector == 0)
 printf "File/Dir not found"
 stop
 else
 Loading the sector
4. while (sector name != name)
 sector = Next_hash sector

```
if (sector == 0)
    printf "File/Dir not found"
else
    Loading the sector
```

The Next_hash field is found in File header, Directory and Link blocks.

Filenames can contain lowercase and uppercase, but the Hash function tells us that 'file1' and 'File1' are a same file.

3.2.2 How to list directory entries ?

The HashTable contains the entries block number.

To obtain the list of a directory entries (files, directories or links) the algorithm is :

```
long entry;
int i;

for(i=0; i<ht_size; i++) {
    entry = HashTable[i]
    while ( entry !=0 ) {
        printf "entry name"
        entry = Next_hash
    }
}
```

3.2.3 How to compute the checksum ?

```
#define Short(p) ((p)[0]<<8 | (p)[1])
#define Long(p) (Short(p)<<16 | Short(p+2))

unsigned long newsum;
unsigned char buf[512];          /* contains rootblock */
int i;

memset(buf+20,0,4);             /* clear old checksum */
```

```

newsum=0L;
for(i=0; i<128; i++)
    newsum+=Long(buf+i*4);
newsum=-newsum;

```

This checksum algorithm works for most blocks type except noted.

The AmigaDos command 'relabel' rename the disk.
The 'format' command is used for ... formatting.

The bitmap table ('bm_pages[]') stores one or several pointers to Bitmap blocks.
The first is at index 0.

3.3 How is managed the free and used blocks list?

Bitmap blocks stores if a sector is free or allocated. One bit is used per sector. If the bit is set, the sector is free, a cleared bit means a allocated sector.

Boot blocks allocation (2 for a floppy disk) is not stored in bitmap. Bitmap is stored in longs, the first sector of a long is the bit 0.

* Bitmap block (512 bytes), often rootblock+1

0/0	long	1	checksum	special algorithm
4/4	long	127	map	

Here follows for a DD disk the relationship between bitmap and sector number :

block #	long #	bit #
2	0	0
3	0	1
4	0	2
...		
33	0	31

34	1	0
35	1	1
...		
880	27	2
881	27	3
...		
1759	54	28
1760	54	29

This map is 1758 bits long (1760-2) and is stored on 54 full long and the first 30th bits of the 55th long.

3.3.1 How to compute bitmap checksum ?

```

#define Short(p) ((p)[0]<<8 | (p)[1])
#define Long(p) (Short(p)<<16 | Short(p+2))

unsigned long newsum;
unsigned char buf[512];          /* contains block */
int i;

newsum=0L;
for(i=1; i<128; i++)            /* ignores old checksum */
    newsum=newsum-Long(buf+i*4);

```

3.3.2 What is the 'bm_ext' field in Rootblock ?

If 25 bitmap blocks (which pointers are stored on Rootblock) are not enough (for Hard Disks), extra needed bitmap blocks pointers are stored in bitmap extension blocks.

* Bitmap extension block (512 bytes) (Hard disk only)

0/0	long	127	bitmap block pointers
508/1fc	long	1	next (0 for last)

The Bitmap extension chained list start at Rootblock with the 'bm_ext'.

3.4 How are stored files on a disk ?

The first block of a file is the File header block, which contains information about the file (size, last access date, ...) and pointers to the first 72th Data blocks.

Data blocks store file data.

If more than 72 Data blocks are needed to store a file, the other Data block pointers are stored in File extension blocks.

File extension blocks are organised in a chained list, which starts in File header block ('extension').

* File header block (512 bytes)

0/0	long	1	type	block primary type T_HEADER (==2)
4	long	1	header_key	self pointer
8	long	1	high_seq	number of data block ptr stored here
12/c	long	1	data_size	unused (==0)
16/10	long	1	first_data	first data block ptr
20/14	long	1	chksum	same algorithm as rootblock
24/18	long	72	data_blocks[]	data blk ptr (first at [high_seq-1])
	...			
320/140	long	1	protect	protection flags (bit set)
				0 delete forbidden (D)
				1 modification forbidden (E)
				2 write/update forbidden (W)
				3 read forbidden (R)
				4 file is an archive (A)
				5 pure, can be made resident (P)
				6 file is a script (S)
				7 hidden file with DIR (H)
324/144	long	1	byte_size	file size
328/148	char	1	comm_len	file comment length
329/149	char	22	comment[]	comment
	...			
420/1a4	long	1	days	last access date (days since 1 jan 1978)
424/1a8	long	1	mins	last access time
428/1ac	long	1	ticks	
432/1b0	char	1	name_len	filename length
433/1b1	char	30	filename[]	filename

```

...
468/1d4 long 1 real_entry FFS : unused (== 0)
472/1d8 long 1 next_link FFS : links chained list (first =
newest)
...
496/1f0 long 1 hash_chain next entry ptr with same hash
500/1f4 long 1 parent parent directory
504/1f8 long 1 extension pointer to 1st file extension block
508/1fc long 1 sec_type secondary type : ST_FILE (== -3)
-----

```

Here we are discovering the first method to read a file : Data block pointers tables. The first table is in File header block, the others in File extension blocks.

For an empty file, a Header file block is written, with 'byte_size' equal to 0, and no Data block pointers in 'data_blocks[]'.

* File extension block (512 bytes) (first pointer in File header)

```

-----
0/0 long 1 type primary type : T_LIST (== 16)
4/4 long 1 header_key self pointer
8/8 long 1 high_seq number of data blk ptr stored
12/c long 1 data_size unused (== 0)
16/10 long 1 first_data unused (== 0)
20/14 long 1 chksum rootblock algorithm
24/18 long 72 data_blocks[] data blk ptr (first at [high_seq-1])
long 45 unused
long 1 info unused (== 0)
long 1 hash_chain; unused (== 0)
500/1f4 long 1 parent ptr to parent directory
504/1f8 long 1 extension next file header extension block, 0
for last
508/1fc long 1 sec_type secondary type : ST_FILE (== -
3)
-----

```

* Data blocks (512 bytes) (first pointer in File header 'first_data' and 'data_blocks[71]')

OFS

```

-----
0/0 long 1 type primary type : T_DATA (== 8)
4/4 long 1 header_key pointer to file header block

```

8/8	long	1	seq_num	file data block number (first is #1)
12/c	long	1	data_size	data size (<= 488)
16/10	long	1	next_data	next data block ptr (0 for last)
20/14	long	1	chksum	rootblock algorithm
24/18	UCHAR	488	data[]	file data

In OFS, there is a second way to read a file : using the Data block chained list.

The list starts in File header ('first_data') and goes on with 'next_data' in each Data block.

FFS

0/0	UCHAR	512	data[]	file data
-----	-------	-----	--------	-----------

In FFS, the only way to read or recover a file is to use File extension blocks. If a File extension block is unreadable, there is no way to find the corresponding Data blocks.

The OFS is more robust than FFS, but slower and can store less data on disk. As you see, disk salvaging is easier with OFS.

When a file is deleted, only its File header block number is cleared from the Directory block (or from the same-hash-value list). File header block, Data blocks and File extension blocks are not cleared ! The undelete operation is easy .

3.5 How are stored directories ?

Directory blocks are very similar to Rootblock, except they don't need information about bitmap and disk, but have comments.

* Directory block (512 bytes)

0/0	long	1	type	primary type : T_HEADER (== 2)
4/4	long	1	header_key	self pointer
8/8	long	1	high_seq	unused (== 0)
12/c	long	1	ht_size	unused (== 0)

	long	1	unused	
20/14	long	1	chksum	normal algorithm
24/18	long	72	ht[]	HashTable (file/dir/link block number)
	long	2	unused	
320/140	long	1	protect	protect flag (like file header)
	long	1	unused	
328/148	char	1	comm_len	comment length
329/149	char	22	comment[]	comment
	char	69	unused	
420/1a4	long	1	days	last access date
424/1a8	long	1	mins	last access time
428/1ac	long	1	ticks	
432/1b0	char	1	name_len	name length
433/1b1	char	30	dirname[]	directory name
	char	5	unused	
468/1d4	long	1	real_entry	unused (== 0)
472/1d8	long	1	next_link	FFS : links chained list
	long	4	unused	
496/1f0	long	1	next_hash	next entry with same hash value
500/1f4	long	1	parent	parent directory
504/1f8	long	1	extension	FFS : first directory cache block
508/1fc	long	1	sec_type	secondary type : ST_UDIR (== 2)

You can obtain directory listing exactly like with the root directory.

3.6 How are implemented links with AmigaDos ?

Only FFS handles links and directory caching.

Soft links are not supported by AmigaDos 3.0. Linked files are seen as directories...

However, some shells (like Csh 5.37) supports them, so i'm supplying the structure.

3.6.1 Hard links

* Hard link (512 bytes)

0/0	long	1	type	T_HEADER = 2
4/4	long	1	self pointer	
	long	3	unused (== 0)	
20/14	long	1	checksum	
	...		unused	

```

420/1a4 long      1      days      date
424/1a8 long      1      mins      time
428/1ac long      1      ticks
432/1b0 char      1      name len
433/1b1 char     30     name[]    link name (dir or file name)
...            unused
468/1d4 long      1      real_entry pointer to real entry
472/1d8 long      1      next_link  (links chained list )
...            unused
500/1f4 long      1      parent    parent directory
...            unused
508/1fc long      1      sec_type  FILE = -4, DIR = 4

```

A 'real' entry is a file or directory entry, opposed to link entries.

A hard link can only be created on the same disk as the real entry disk. Several links can be made on the same real entry. This is stored with a chained list.

'real entry' always contains the real entry block pointer.

'next_link' stores the links chained list.

Adding are made at head :

```

>ls
-----rw-d      1912  15-May-96 22:28:08  real

```

Chained list state :

```

block# real      next      name
-----
484      0              0        real

```

>ln real link1

```

>ls
-----rw-d      1912  15-May-96 22:28:08  real
-H----rw-d      1912  15-May-96 22:28:10  link1 -> Empty:real

```

```

block# real      next      name
-----
484      0              104     real
104      484            0        link1

```

>ln link1 link2

>ls

```

-----rw-d      1912  15-May-96  22:28:08  real
-H----rw-d      1912  15-May-96  22:28:10  link1 -> Empty:real
-H----rw-d      1912  15-May-96  22:28:12  link2 -> Empty:real

```

```

block#  real      next      name
-----
484     0             107     real
104     484           0       link1
107     484           104     link2

```

The links are stored 'newest first', due to the adding at head.

real -> newest link -> ... -> oldest link -> 0

-> means "point to"

3.6.2 Soft links

* Soft link (512 bytes)

```

-----
0/0     long      1         type           T_HEADER (== 2)
4/4     long      1         self pointer
...
20/14   long      1         checksum
24/18   char      30        real entry name (doesn't need to exist !)
...
420/1a4 long      1         days           creation date (access is real's)
424/1a8 long      1         mins
428/1ac long      1         ticks
...
500/1f4 long      1         parent        parent directory
...
508/1fc long      1         sec_type      ST_LSOFT (== 3)
-----

```

3.7 How is the block associated to directory caching ?

To speed up directory listing, Directory cache blocks have been created. Directory cache blocks are also organised in chained lists.

The list starts at the directory block (root or normal directory) with 'extension'.

* Directory cache block (512 bytes)

0/0	long	1	type	DIRCACHE == 33 (0x21)
4/4	long	1	header_key	self pointer
8/8	long	1	parent	parent directory
12/c	long	1	records_nb	directory entry records in this block
16/10	long	1	next_dirc	dir cache chained list
20/14	long	1	chksum	normal checksum
24/18	UCHAR	488	records[]	entries list

The directory entries are stored this way :

* Directory cache block entry record (26 <= size (in bytes) <= 77)

0	long	1	header	entry block pointer
4	long	1	size	file size (0 for a directory)
8	long	1	protect	protection flags
12	long	1	unused	(== 0)
16	short	1	days	date
18	short	1	mins	time
20	short	1	ticks	
22	char	1	type	secondary type
23	char	1	name_len	>=1, <=30
?	char	?	name	name
?	char	1	comm_len	>=0, <=22
?	char	?	comment	comment
?	(char)	0	optional padding byte	(680x0 longs must be word aligned)

4. What is a blank disk ?

=====

A minimal blank disk has a Bootblock, a Rootblock and a Bitmap block.

4.1 a Minimal blank disk

* The Bootblock (0 and 1)

0	char	4	ID	'D''O''S' + flags (0 -> 5)
4	long	1023	full of zeros	

* The Rootblock (880)

0	long	1	type	2
12/c	long	1	ht_size	0x48
20/14	long	1	checksum	computed
312/138	long	1	bm_flag	-1 (valid bitmap)
316/13c	long	1	bm_pages[0]	bitmap sector #
432/1b0	char	1	disk_name size	?
433/1b1	char	?	disk_name	?
472/1d8	long	1	last access date	
476/1dc	long	1	last access time	
480/1e0	long	1	last access time	
484/1e4	long	1	creation date	
488/1e8	long	1	creation time	
492/1ec	long	1	creation time	
504/1f8	long	1	FFS : first dir cache sector	or 0
508/1fc	long	1	sub_type	1

No specified fields are null.

* The Bitmap block (here 881) for a DD disk

0	long	1	checksum	
4	long	27	free sectors	0xffffffff
112/70	long	1	root+bitmap	0xffff3fff
116/74	long	27	free sectors	0xffffffff
120/78	long	72	unused	!=0

4.2 A 'Bootable' disk

* The Bootblock becomes :

0	long	1	ID	'D''O''S' + flags
4	long	1	checksum	computed
8	long	1	rootblock ?	880
12/c	byte	81	bootcode	AmigaDos 3.0 version

values

disassembled

```

-----+-----
43FA003E          lea    exp(pc),a1      ;Lib name
7025             moveq  #37,d0       ;Lib version
4EAEFDD8        jsr    -552(a6)      ;OpenLibrary()
4A80            tst.l  d0              ;error == 0
670C           beq.b  else
2240           move.l  d0,a1      ;lib pointer
08E90006 0022    bset  #6,34(a1)      ; ?
4EAEFE62        jsr    -414(a6)      ;CloseLibrary()
43FA0018      else:  lea    dos(PC),a1    ;name
4EAEFFA0        jsr    -96(a6)      ;FindResident()
4A80            tst.l  d0
670A           beq.b  else2      ;not found
2040           move.l  d0,a0
20680016       move.l  22(a0),a0    ;DosInit sub
7000           moveq  #0,d0
4E75           rts
70FF      else2:  moveq  #-1,d0
4E75           rts
646F732E 6C696272 617279
                dos:   "dos.library"
00
                ;padding byte
65787061 6E73696F 6E2E6C69 62726172 79
                exp:   "expansion.library"

```

93/5d byte 931 full of zeros

4.3 A Directory cache mode disk

* A directory cache block (882)

0	long	1	type	0x21
4	long	1	self pointer	882
8	long	1	cached dir	880 (root)
12/c	long	1	entries number	0
16/10	long	1	next dir cache	0 (last)
20/14	long	1	checksum	computed
24	long	122	full of zeros	

5. References

=====

* ASM Sources:

Scoopex and Crionics disassembled demo hardloaders
'the floppy disk book' copier source file, DATA BECKER books, 1988

* On-Line material :

- Very good 'ded.doc' file including Hard Disk information :
<ftp://ftp.funet.fi/pub/amiga/utilities/disk/Ded-1.11.lha>
- A clean track-loader which don't use AmigaDOS :
<ftp://ftp.wustl.edu/pub/aminet/dev/asm/t-loader.lha>
- A replacement for 'trackdisk.device' :
<ftp://ftp.wustl.edu/pub/aminet/disk/misc/hackdisk202.lha>
- 'amigadisk_hardware.doc' (by Dave Edwards, Mark Meany, ... of ACC)
<http://home.sol.no/svjohan/assem/refs/diskHW.lha>

* Books :

Rom Kernel Reference Manual : Hardware, pages 235-244, Addison Wesley
La Bible de l'Amiga, Dittrich/Gelfand/Schemmel, Data Becker, 1988.

The AmigaDos reference manual must contains a lot of information about Amiga file systems, but i don't own it (Addison Wesley).

6. C Routines

=====

Unix system routines yet implemented :

cd, pwd, getdent, ls, ls -l, ls -r, open, close, read, stat.

newfs, mount, umount.

Links and directory cache are supported.

Soon available on good FTP servers.

7. Other Amiga FileSystems

=====

An Amiga filesystem for Linux 0.99pl2 by Ray Burr (read only, hard disk support) :

<ftp://tsx-11.mit.edu/pub/linux/patches/amigaffs.tar.Z>

An enhanced version for Linux 1.2.13 by Hans-Joachim "JBHR" Widmaier (RDSK,

links and

international mode supported):

<<ftp://ftp.ibp.fr/pub/linux/sunsite/system/filesystems/jb-affs-1.0.tar.gz>>

A .ADF manipulation package for DOS/Windows, "ADF-suite" (GUI, no sources included):

<<http://www.geocities.com/SiliconValley/Lakes/7094/index.html>>

8. To do

=====

- Hard disk support
- Disk salvaging
- Undelete files
- ADF monitor/editor

'FAQ : THE AMIGA FLOPPY DISK FORMAT' ends here !