HFS Utilities

Complete Technical Manual

HFS and HFS+ Filesystem Utilities for Unix Systems

hfsutils Project

December 17, 2025

Version 4.1.0A.2

Copyright and License

Contents

1 Filesystem History and Evolution **6**
- 1.1 Overview . . . 6
- 1.2 Macintosh File System (MFS) . . . 6
  - 1.2.1 Introduction . . . 6
  - 1.2.2 Key Characteristics . . . 6
  - 1.2.3 Design Philosophy . . . 6
  - 1.2.4 Limitations . . . 7
  - 1.2.5 MFS Structure Summary . . . 7
- 1.3 Hierarchical File System (HFS) . . . 7
  - 1.3.1 Introduction . . . 7
  - 1.3.2 Key Improvements over MFS . . . 7
  - 1.3.3 Timeline . . . 7
  - 1.3.4 HFS Characteristics . . . 8
- 1.4 HFS Plus (HFS+) . . . 8
  - 1.4.1 Introduction . . . 8
  - 1.4.2 Major Enhancements . . . 8
  - 1.4.3 Timeline . . . 8
  - 1.4.4 HFS+ Variants . . . 9
  - 1.4.5 HFS+ Characteristics . . . 9
- 1.5 Apple File System (APFS) . . . 9
  - 1.5.1 Introduction . . . 9
  - 1.5.2 Design Goals . . . 10
  - 1.5.3 Key Features . . . 10
  - 1.5.4 APFS vs HFS+ . . . 10
  - 1.5.5 APFS Adoption . . . 10
  - 1.5.6 Why HFS+ Still Matters . . . 11
- 1.6 Filesystem Evolution Summary . . . 11
- 1.7 Scope of This Document . . . 11
- 1.8 Historical Oddities and Notes . . . 11
  - 1.8.1 The ”BD” Signature Mystery . . . 11
  - 1.8.2 The 1904 Epoch . . . 12
  - 1.8.3 The Colon as Path Separator . . . 12
  - 1.8.4 The 31 vs 255 Character Limit . . . 12
  - 1.8.5 Case Sensitivity Confusion . . . 12

2 Introduction **13**
- 2.1 Overview . . . 13
  - 2.1.1 Purpose and Goals . . . 13
  - 2.1.2 Supported Systems . . . 13
  - 2.1.3 Key Features . . . 14
- 2.2 Installation . . . 15

2.2.1 Building from Source . . . 15
2.2.2 Verifying Installation . . . 15
2.3 Quick Start . . . 16
2.3.1 Creating a Filesystem . . . 16
2.3.2 Checking a Filesystem . . . 16
2.3.3 Mounting a Filesystem . . . 16
2.3.4 Using hfsutil Commands . . . 17
2.4 Documentation Structure . . . 17

3 HFS Specification **18**
3.1 HFS Filesystem Overview . . . 18
3.1.1 Key Characteristics . . . 18
3.1.2 Volume Structure . . . 18
3.2 Master Directory Block (MDB) - Complete Specification . . . 19
3.2.1 MDB Complete Field Map - Every Byte Documented . . . 19
3.2.2 Critical Field Details - Bit-by-Bit . . . 20
3.2.3 Extent Records - Complete Structure . . . 22
3.2.4 Alternate MDB - Critical Backup . . . 22
3.3 HFS B-Trees - Complete Specification . . . 23
3.3.1 B-Tree Overview . . . 23
3.3.2 Node Descriptor - First 14 Bytes of Every Node . . . 23
3.3.3 Header Node (Node 0) - Complete Structure . . . 23
3.3.4 Date/Time Representation . . . 24
3.4 Byte Order (Endianness) - Critical . . . 24
3.4.1 Endianness Examples . . . 24
3.5 Oddities, Edge Cases, and Implementation Notes . . . 25
3.5.1 The 16-Bit Limitation . . . 25
3.5.2 Pascal Strings vs C Strings . . . 25
3.5.3 Allocation Block Alignment . . . 25
3.5.4 MacRoman Character Encoding . . . 26

4 HFS+ Specification **27**
4.1 HFS+ Filesystem Overview . . . 27
4.1.1 Key Improvements over HFS . . . 27
4.1.2 HFS+ Characteristics . . . 27
4.2 Volume Structure . . . 28
4.2.1 Volume Layout . . . 28
4.3 Volume Header - Complete Bit-Level Specification . . . 28
4.3.1 Volume Header Complete Field Map - Every Byte Documented . . . 28
4.3.2 Critical Field Details - Bit-by-Bit . . . 29
4.3.3 HFSPlusForkData Structure - 80 Bytes Per Fork . . . 32
4.3.4 Alternate Volume Header Location . . . 32
4.4 HFS+ B-Trees - Complete Node and Record Structures . . . 33
4.4.1 B-Tree Node Structure - 14-Byte Node Descriptor . . . 33
4.4.2 BTHeaderRec - B-Tree Header Record (106 Bytes) . . . 33
4.4.3 Catalog File B-Tree - Complete Key and Record Formats . . . 35
4.4.4 Catalog File . . . 36
4.4.5 Extents Overflow File . . . 37
4.4.6 Attributes File . . . 38
4.5 Unicode Normalization - CRITICAL for Filename Compatibility . . . 38
4.5.1 NFD vs NFC - The Core Problem . . . 38
4.5.2 HFS+ NFD Requirement - MANDATORY . . . 39

4.5.3 Common NFD Characters - Complete Table . . . 39
4.5.4 Compatibility Issues . . . 39
4.6 HFS+ Time Format - Mac Epoch and Conversion . . . 40
4.6.1 Mac Epoch Definition . . . 40
4.6.2 Date Range . . . 40
4.6.3 Conversion Formulas . . . 40
4.6.4 Byte Representation . . . 40
4.6.5 Y2K40 Safeguards in hfsutils . . . 41
4.7 HFS+ Critical Oddities and Edge Cases . . . 41
4.7.1 Case-Insensitive vs Case-Preserving . . . 41
4.7.2 Folder Valence - Hidden Complexity . . . 41
4.7.3 Hard Links - Indirect Nodes . . . 42
4.7.4 Compression - Undocumented Extension . . . 42
4.7.5 Journal Checksum Algorithm - Missing from TN1150 . . . 42
4.7.6 Allocation Block Alignment . . . 42
4.7.7 Extended Attributes File - Optional . . . 43
4.8 Reimplementation Checklist - Everything You Need . . . 43
4.8.1 Data Structures Required . . . 43
4.8.2 Algorithms Required . . . 43
4.8.3 Validation Commands . . . 44
4.8.4 No External References Needed . . . 44
4.9 HFS+ vs HFSX . . . 44
4.10 Compatibility Considerations . . . 45
4.10.1 macOS . . . 45
4.10.2 Linux . . . 45
4.10.3 FreeBSD/OpenBSD/NetBSD . . . 45
4.10.4 Windows . . . 45

5 mkfs Utilities **46**
5.1 mkfs.hfs - HFS Filesystem Creation . . . 46
5.1.1 Command Synopsis . . . 46

6 fsck Utilities **47**
6.1 fsck.hfs - HFS Filesystem Check . . . 47
6.1.1 Command Synopsis . . . 47

7 mount Utilities **48**
7.1 mount.hfs - HFS Filesystem Mounting . . . 48
7.1.1 Command Synopsis . . . 48

8 hfsutil Commands **49**
8.1 hfsutil Commands . . . 49
8.1.1 Commands . . . 49

9 Implementation Details **50**
9.1 Implementation Details . . . 50
9.1.1 Topics to Cover . . . 50

10 Testing and Validation **51**
10.1 Testing and Validation . . . 51
10.1.1 Zero-Tolerance Policy . . . 51
10.1.2 Test Suite . . . 51

A Appendix **52**
A.1 Structure Definitions 52
A.2 Error Codes 52
A.3 Glossary 52
A.4 References 52

Chapter 1

Filesystem History and Evolution

1.1 Overview

This chapter traces the evolution of Apple's filesystems from the original Macintosh File System (MFS) through HFS, HFS+, and the modern APFS. Understanding this evolution provides context for design decisions and compatibility requirements.

1.2 Macintosh File System (MFS)

1.2.1 Introduction

MFS (Macintosh File System) was the original filesystem used on the Macintosh 128K and 512K (1984-1985). It was designed for floppy disks and small hard drives.

1.2.2 Key Characteristics

- **Introduced**: January 1984 (Macintosh 128K)
- **Maximum volume size**: 20 MB (limited by hardware)
- **Maximum files**: 65,535 (16-bit file numbers)
- **Filename length**: 255 characters (Macintosh Roman)
- **Directory structure**: Flat (no folders/subdirectories)
- **Dual forks**: Data fork and resource fork
- **File types**: 4-character type and creator codes

1.2.3 Design Philosophy

MFS was revolutionary for its time:

- First consumer filesystem with resource forks
- Integrated with Finder for desktop metaphor
- Type/creator codes enabled double-click launching
- Desktop file tracked file positions and icons

1.2.4 Limitations

- **No hierarchy**: All files stored in single directory
- **Finder folders**: Simulated by Finder, not filesystem
- **Small volumes**: Designed for 400K floppies
- **Poor scalability**: Performance degraded with many files

1.2.5 MFS Structure Summary

Component	Description
Logical Block 0-1	Boot blocks (1024 bytes)
Logical Block 2	Master Directory Block (MDB)
Following blocks	File directory (single B-tree)
...	Allocation bitmap
...	File data
Last sector	Alternate MDB

Table 1.1: MFS Volume Layout

Note: MFS is not implemented in hfsutils but documented for historical context and understanding HFS evolution.

1.3 Hierarchical File System (HFS)

1.3.1 Introduction

HFS (Hierarchical File System) replaced MFS in Mac OS System 2.1 (1985) and System 3.0 (1986). It introduced true hierarchical directories.

1.3.2 Key Improvements over MFS

- **Hierarchical directories**: True folder structure
- **Larger volumes**: Up to 2 GB (later 2 TB with HFS wrapper)
- **Better performance**: Separate B-trees for catalog and extents
- **Allocation optimization**: Clump sizes reduce fragmentation

1.3.3 Timeline

- **1985**: Introduced with Hard Disk 20
- **1986**: Became standard in System 3.0
- **1998**: Superseded by HFS+ in Mac OS 8.1
- **1999**: Last use in Mac OS 9 for booting
- **2000s**: Maintained for compatibility
- **Today**: Supported for legacy media

1.3.4 HFS Characteristics

Detailed in Chapter 2 (HFS Specification). Summary:

Feature	Specification
Signature	0x4244 ('BD')
Maximum volume	2 TB
Maximum file	2 GB
Filename length	31 characters (MacRoman)
Allocation blocks	16-bit addressing (65,535 blocks)
B-tree node size	512 bytes (fixed)
Date range	1904-2028 (Y2K28 limit)
Case sensitivity	Case-insensitive, case-preserving

Table 1.2: HFS Characteristics

1.4 HFS Plus (HFS+)

1.4.1 Introduction

HFS+ (HFS Plus, "Mac OS Extended") was introduced in Mac OS 8.1 (1998) to address HFS limitations for modern computing.

1.4.2 Major Enhancements

- **Unicode filenames**: UTF-16, up to 255 characters
- **32-bit addressing**: Support for very large volumes
- **Smaller allocation blocks**: Better space efficiency
- **Journaling**: Optional crash recovery (Mac OS X 10.2.2+)
- **Hard links**: Unix-style hard links
- **Symbolic links**: Unix-style symbolic links
- **Extended attributes**: Arbitrary metadata
- **HFSX variant**: Case-sensitive option

1.4.3 Timeline

- **1998**: Introduced in Mac OS 8.1
- **1999**: Became default in Mac OS 8.6
- **2001**: Mac OS X adoption
- **2002**: Journaling added (Mac OS X 10.2.2)
- **2005**: Case-sensitive HFSX variant
- **2017**: Superseded by APFS (macOS 10.13)
- **Today**: Still widely used, especially on spinning drives

1.4.4 HFS+ Variants

Standard HFS+

- Signature: 0x482B ('H+')
- Version: 4
- Case-insensitive filenames
- Most compatible

HFSX (HFS Extended)

- Signature: 0x4858 ('HX')
- Version: 5
- Case-sensitive filenames
- Used for Unix-like behavior

Journaled HFS+

- Attributes bit 13 (0x2000) set
- Journal info block pointer in Volume Header
- Circular journal buffer
- **Not supported by Linux kernel**

1.4.5 HFS+ Characteristics

Detailed in Chapter 3 (HFS+ Specification). Summary:

Feature	Specification
Signature	0x482B ('H+') or 0x4858 ('HX')
Maximum volume	8 EB theoretical
Maximum file	8 EB theoretical
Filename length	255 UTF-16 characters
Allocation blocks	32-bit addressing
B-tree node size	4096 bytes (typical)
Date range	1904-2040 (Y2K40 limit)
Case sensitivity	Optional (HFSX)
Journaling	Optional

Table 1.3: HFS+ Characteristics

1.5 Apple File System (APFS)

1.5.1 Introduction

APFS (Apple File System) is Apple's modern filesystem, introduced in macOS 10.13 High Sierra (2017). It replaces HFS+ as the default for SSDs and flash storage.

1.5.2 Design Goals

- **Flash optimized**: Minimize write amplification
- **Space sharing**: Multiple volumes share space pool
- **Snapshots**: Instant, space-efficient snapshots
- **Cloning**: Copy-on-write file/directory clones
- **Strong encryption**: Native full-disk and per-file encryption
- **Crash protection**: Copy-on-write metadata

1.5.3 Key Features

- 64-bit inode numbers
- Nanosecond timestamp precision
- Space sharing between volumes
- Native encryption (FileVault)
- Fast directory sizing
- Atomic safe-save operations
- Clones and snapshots

1.5.4 APFS vs HFS+

Feature	**HFS+**	**APFS**
Introduced	1998	2017
Optimized for	HDDs	SSDs/Flash
Snapshots	No	Yes
Cloning	No	Yes (instant)
Encryption	Per-volume	Per-file + volume
Space sharing	No	Yes
Timestamps	Second precision	Nanosecond precision
Max files	4 billion	9 quintillion
Journaling	Optional	Always (COW)

Table 1.4: HFS+ vs APFS Comparison

1.5.5 APFS Adoption

- **macOS 10.13+**: Default for SSDs
- **iOS 10.3+**: Default for all devices
- **watchOS 4+**: Default
- **tvOS 11+**: Default
- **HDDs**: HFS+ still recommended (as of macOS 13)

1.5.6 Why HFS+ Still Matters

- **Legacy systems**: Pre-2017 Macs
- **HDDs**: APFS not optimized for spinning drives
- **External drives**: Better compatibility
- **Recovery partitions**: Some use HFS+
- **Boot Camp**: Windows compatibility
- **Linux support**: Better HFS+ kernel support

1.6 Filesystem Evolution Summary

FS	Year	Max Volume	Max File	Key Innovation
MFS	1984	20 MB	N/A	Resource forks
HFS	1985	2 TB	2 GB	Hierarchy
HFS+	1998	8 EB	8 EB	Unicode, large files
APFS	2017	8 EB	8 EB	Flash-optimized, snapshots

Table 1.5: Apple Filesystem Evolution

1.7 Scope of This Document

This manual focuses on:

- **HFS (Classic)**: Complete implementation (Chapter 2)
- **HFS+**: Complete implementation (Chapter 3)
- **MFS**: Historical context only
- **APFS**: Overview for migration context

All specifications are documented at the bit level to enable complete reimplementation without external references.

1.8 Historical Oddities and Notes

1.8.1 The "BD" Signature Mystery

HFS uses signature 0x4244 ('BD' in ASCII). The origin is unclear:

- Possibly "Block Device"
- Possibly arbitrary choice
- Never officially documented by Apple

1.8.2 The 1904 Epoch

All Apple filesystems use January 1, 1904 as time zero:

- Predates Unix epoch (1970) by 66 years
- Reason unknown but consistent across all Apple filesystems
- Creates Y2K28 (HFS) and Y2K40 (HFS+) problems

1.8.3 The Colon as Path Separator

Classic Mac OS used colon (:) as path separator:

- Unix/Windows use / and \respectively
- Filenames cannot contain colons
- Causes issues when sharing files cross-platform
- Mac OS X translates : to / for display

1.8.4 The 31 vs 255 Character Limit

- HFS: 31 characters (historical, tied to original Mac)
- HFS+: 255 characters (modern standard)
- MFS: Actually 255 characters (more than HFS!)

1.8.5 Case Sensitivity Confusion

- HFS/HFS+: Case-insensitive by default
- HFSX: Case-sensitive variant
- Linux ext4: Case-sensitive
- Windows NTFS: Case-insensitive but preserving
- Source of many cross-platform file issues

Chapter 2

Introduction

2.1 Overview

The HFS Utilities (hfsutils) project provides a comprehensive set of tools for working with Apple's HFS (Hierarchical File System) and HFS+ (HFS Plus) filesystems on Unix-like operating systems including Linux, BSD, and macOS.

2.1.1 Purpose and Goals

HFS and HFS+ filesystems are commonly used on:

- Classic Mac OS systems (HFS)
- Mac OS X and macOS systems prior to APFS (HFS+)
- iPod devices
- External drives formatted for Mac compatibility
- Disk images and backups from Apple systems

This toolset enables Unix systems to:

- **Create** HFS and HFS+ filesystems with `mkfs.hfs` and `mkfs.hfs+`
- **Check and repair** filesystem integrity with `fsck.hfs` and `fsck.hfs+`
- **Mount** HFS and HFS+ volumes with `mount.hfs` and `mount.hfs+` (requires kernel support)
- **Manipulate files** on HFS volumes without mounting using `hfsutil` commands (useful on systems without HFS kernel drivers)

2.1.2 Supported Systems

The utilities work on any POSIX-compliant system with appropriate kernel support:

Note: On systems without kernel HFS support, the `hfsutil` commands can still be used to access HFS filesystem contents.

System	HFS Support	HFS+ Support
Linux	Kernel module `hfs`	Kernel module `hfsplus`
FreeBSD	Native support	Native support
macOS	Native support	Native support
OpenBSD	Via FUSE	Via FUSE
NetBSD	Via FUSE	Via FUSE

Table 2.1: Platform Support Matrix

2.1.3 Key Features

Full Specification Compliance

All utilities strictly adhere to:

- Inside Macintosh: Files (HFS specification)
- Apple Technical Note TN1150 (HFS+ specification)
- Unix/POSIX filesystem utility standards

Zero-Tolerance Validation

The test suite implements a "zero-tolerance" policy:

- ANY deviation from specification = test failure
- ALL filesystems must be 100% correct
- Complete validation before and after fsck operations

Journaling Support

HFS+ journaling is supported with appropriate warnings:

- Journal creation with `mkfs.hfs+ -j`
- Journal validation and replay in `fsck.hfs+`
- Linux kernel compatibility warnings (journaling not supported in Linux HFS+ driver)

Date Limit Awareness

The utilities handle filesystem date limits correctly:

- HFS: Maximum date February 6, 2028 (Y2K28 problem)
- HFS+: Maximum date February 6, 2040 (Y2K40 problem)
- Automatic date correction to safe values

2.2 Installation

2.2.1 Building from Source

Prerequisites

```
# Debian/Ubuntu
sudo apt-get install build-essential

# Fedora/RHEL
sudo dnf install gcc make

# macOS
xcode-select --install

# BSD
# Compiler included by default
```

Compilation

```
# Clone repository
git clone https://github.com/JotaRandom/hfsutils.git
cd hfsutils

# Build all utilities
make

# Build specific sets
make set-hfs          # mkfs.hfs, fsck.hfs, mount.hfs
make set-hfsplus      # mkfs.hfs+, fsck.hfs+, mount.hfs+

# Build with hfsutil
make all
```

Installation Options

Linux systems (with kernel HFS/HFS+ drivers):

```
sudo make install-linux PREFIX=/usr
```

Complete installation (filesystem utilities + hfsutil):

```
sudo make install-complete PREFIX=/usr/local
```

Individual utilities:

```
sudo make install-mkfs.hfs+ PREFIX=/usr
sudo make install-fsck.hfs PREFIX=/usr
sudo make install-mount.hfs+ PREFIX=/usr
```

2.2.2 Verifying Installation

After installation, verify the utilities are available:

```
mkfs.hfs --version
mkfs.hfs+ --version
fsck.hfs --version
```

```
fsck.hfs+ --version
mount.hfs --help
mount.hfs+ --help
hfsutil --version   # If installed
```

Check manpages:

```
man mkfs.hfs
man mkfs.hfs+
man fsck.hfs+
man mount.hfs+
```

2.3 Quick Start

2.3.1 Creating a Filesystem

Create an HFS filesystem:

```
# Create 10MB image file
dd if=/dev/zero of=test.img bs=1M count=10

# Format as HFS
mkfs.hfs -L "MyDisk" test.img
```

Create an HFS+ filesystem:

```
# Create 50MB image file
dd if=/dev/zero of=test.img bs=1M count=50

# Format as HFS+
mkfs.hfs+ -L "MyDisk" test.img

# Format with journaling (Linux warning will appear)
mkfs.hfs+ -j -L "MyDisk" test.img
```

2.3.2 Checking a Filesystem

Verify filesystem integrity:

```
# Check HFS filesystem (read-only)
fsck.hfs -n test.img

# Check and repair HFS+ filesystem
fsck.hfs+ -y test.img

# Verbose output
fsck.hfs+ -v -n test.img
```

2.3.3 Mounting a Filesystem

On systems with kernel HFS/HFS+ support:

```
# Create mount point
sudo mkdir /mnt/hfs

# Mount read-write
sudo mount.hfs+ test.img /mnt/hfs
```

```

# Mount read-only
sudo mount.hfs+ -r test.img /mnt/hfs

# Unmount
sudo umount /mnt/hfs
```

2.3.4 Using hfsutil Commands

On systems without kernel support or for direct access:

```
# Format volume
hfsutil hformat -L "MyDisk" test.img

# Mount in hfsutil
hfsutil hmount test.img

# List contents
hfsutil hls

# Copy file into volume
hfsutil hcopy myfile.txt :myfile.txt

# Copy file out of volume
hfsutil hcopy :myfile.txt retrieved.txt

# Unmount
hfsutil humount
```

2.4 Documentation Structure

This manual is organized as follows:

Chapter 2: HFS Specification Details of the classic HFS filesystem format

Chapter 3: HFS+ Specification Details of the HFS+ filesystem format with journaling

Chapter 4: mkfs Utilities Filesystem creation tools

Chapter 5: fsck Utilities Filesystem checking and repair tools

Chapter 6: mount Utilities Filesystem mounting tools

Chapter 7: hfsutil Commands File manipulation without mounting

Chapter 8: Implementation Details Internal architecture and algorithms

Chapter 9: Testing and Validation Test suite and quality assurance

Chapter 10: Appendix Structure definitions, error codes, glossary

Chapter 3

HFS Specification

3.1 HFS Filesystem Overview

The Hierarchical File System (HFS) is the filesystem used by Apple Computer for Mac OS systems from 1985 until Mac OS X. Also known as "Mac OS Standard" or "HFS Classic", it provides a hierarchical directory structure with support for file and folder metadata.

3.1.1 Key Characteristics

- **Maximum volume size**: 2 TB (2,199,023,255,552 bytes)
- **Maximum file size**: 2 GB (2,147,483,647 bytes)
- **Filename length**: 31 bytes (Macintosh Roman encoding)
- **Date range**: January 1, 1904 to February 6, 2028 (Y2K28 limit)
- **Allocation block size**: 512 bytes minimum, 64 KB maximum
- **Maximum allocation blocks**: 65,535 (16-bit addressing)
- **Case sensitivity**: Case-insensitive, case-preserving
- **Byte order**: Big-endian (MSB first)

3.1.2 Volume Structure

An HFS volume is divided into **logical blocks** (512 bytes each) and **allocation blocks** (multiples of logical blocks).

Complete Volume Layout

Offset (bytes)	Size	Description
0	1024	Boot blocks (2 logical blocks)
1024	512	Master Directory Block (MDB)
1536	Variable	Allocation bitmap start
...	Variable	Extents B-tree file
...	Variable	Catalog B-tree file
...	Variable	File data area
volume_size - 1536	512	(Reserved space)
volume_size - 1024	512	Alternate MDB

Offset (bytes)	Size	Description
volume_size - 512	512	(Last logical block)

Table 3.1: HFS Physical Volume Layout

Critical Formula for Alternate MDB:

$$AlternateMDBoffset = total\_bytes - 1024 \tag{3.1}$$

This is **NOT** `(total_sectors - 2) * 512`. It is literally 1024 bytes before the end, regardless of sector size.

3.2 Master Directory Block (MDB) - Complete Specification

The MDB is the **single most critical structure** in HFS. It is located at byte offset 1024 and is exactly 162 bytes long.

3.2.1 MDB Complete Field Map - Every Byte Documented

Offset	Field Name	Type	Bytes	Description
+0	drSigWord	uint16	2	Signature: **0x4244** ('BD')
+2	drCrDate	uint32	4	Creation date (Mac time)
+6	drLsMod	uint32	4	Last modification date
+10	drAtrb	uint16	2	Volume attributes (see below)
+12	drNmFls	uint16	2	Files in root directory
+14	drVBMSt	uint16	2	First allocation bitmap block
+16	drAllocPtr	uint16	2	Start of next allocation search
+18	drNmAlBlks	uint16	2	Number of allocation blocks
+20	drAlBlkSiz	uint32	4	Allocation block size (bytes)
+24	drClpSiz	uint32	4	Default clump size
+28	drAlBlSt	uint16	2	First allocation block
+30	drNxtCNID	uint32	4	Next Catalog Node ID
+34	drFreeBks	uint16	2	Free allocation blocks
+36	drVN	Pstring	28	Volume name (1 len + 27 chars)
+64	drVolBkUp	uint32	4	Last backup date
+68	drVSeqNum	uint16	2	Backup sequence number
+70	drWrCnt	uint32	4	Volume write count
+74	drXTClpSiz	uint32	4	Extents clump size
+78	drCTClpSiz	uint32	4	Catalog clump size
+82	drNmRtDirs	uint16	2	Directories in root
+84	drFilCnt	uint32	4	Total files on volume
+88	drDirCnt	uint32	4	Total directories
+92	drFndrInfo	uint32[8]	32	Finder information
+124	drVCSize	uint16	2	Volume cache size
+126	drVBMCSize	uint16	2	Bitmap cache size
+128	drCtlCSize	uint16	2	Common cache size
+130	drXTFlSize	uint32	4	Extents file size
+134	drXTExtRec	ExtRec[3]	12	Extents file extents

Offset	Field Name	Type	Bytes	Description
+146	drCTFlSize	uint32	4	Catalog file size
+150	drCTExtRec	ExtRec[3]	12	Catalog file extents

Table 3.2: Master Directory Block Complete Structure (162 bytes total)

3.2.2 Critical Field Details - Bit-by-Bit

drSigWord - Signature (Offset +0, 2 bytes)

Value: `0x4244` (big-endian)

Byte representation:

```
Offset 1024: 0x42 ('B')
Offset 1025: 0x44 ('D')
```

Validation:

- Must be exactly `0x4244`
- Any other value = not HFS or corrupted
- Endianness test: if you read `0x4442`, you're reading little-endian

Oddity: The origin of "BD" is unknown. Speculation includes "Block Device" but Apple never documented it.

drAtrb - Volume Attributes (Offset +10, 2 bytes)

16-bit flags field (big-endian). **Every bit documented**:

Bit	Hex Mask	Meaning
0-6	0x007F	Reserved (must be 0)
7	0x0080	Volume locked by hardware
8	**0x0100**	**Volume unmounted properly**
9	0x0200	Volume has spared bad blocks
10	0x0400	Volume needs consistency check (kNeedRebuild)
11	0x0800	Catalog node IDs reused (kBadCNID)
12	0x1000	Unused nodes fix needed
13	0x2000	Volume journaled (HFS+ only, not used in HFS)
14	0x4000	Software lock
15	0x8000	Spare boot blocks (not used)

Table 3.3: drAtrb Bit Definitions

Critical: Bit 8 (0x0100) MUST be set for clean unmount. mkfs.hfs sets:

```
drAtrb = 0x0100  // Big-endian bytes: 0x01 0x00
```

Hex dump verification:

```
xxd -s 1034 -l 2 -p volume.hfs
Expected output: 0100
```

drNxtCNID - Next Catalog Node ID (Offset +30, 4 bytes)

Value: 32-bit unsigned integer, big-endian
Minimum: `0x00000010` (16 decimal)
Reserved CNIDs 1-15:

CNID	Purpose
1	Parent of root directory (kHFSRootParentID)
2	Root directory (kHFSRootFolderID)
3	Extents overflow file (kHFSExtentsFileID)
4	Catalog file (kHFSCatalogFileID)
5	Bad allocation blocks file (kHFSBadBlock-FileID)
6-15	Reserved, not used in HFS

Table 3.4: Reserved Catalog Node IDs

Byte representation for value 16:

```
Offset 1054: 0x00
Offset 1055: 0x00
Offset 1056: 0x00
Offset 1057: 0x10
```

Common error: If you see `0x00000000`, the MDB was not initialized correctly. The volume cannot be used.

drVN - Volume Name (Offset +36, 28 bytes)

Format: Pascal string (length-prefixed)

- Byte 0: Length (0-27)
- Bytes 1-27: Characters (Macintosh Roman encoding)

Example: "MyDisk"

```
Offset 1060: 0x06          // Length = 6
Offset 1061: 0x4D ('M')
Offset 1062: 0x79 ('y')
Offset 1063: 0x44 ('D')
Offset 1064: 0x69 ('i')
Offset 1065: 0x73 ('s')
Offset 1066: 0x6B ('k')
Offset 1067-1087: 0x00     // Padding
```

Restrictions:

- Length: 1-27 bytes (0 = invalid)
- Cannot contain colon (:) - path separator
- Macintosh Roman encoding (not UTF-8!)

Oddity: Unlike HFS+, HFS uses Pascal strings (length prefix) instead of C strings (null-terminated).

3.2.3 Extent Records - Complete Structure

An extent record describes up to 3 contiguous runs of allocation blocks.

Extent Descriptor (4 bytes each)

Offset	Field	Description
+0	startBlock	First allocation block (uint16)
+2	blockCount	Number of blocks (uint16)

Table 3.5: Extent Descriptor Structure

Extent Record (12 bytes)

3 consecutive extent descriptors:

```
ExtentRecord {
    ExtentDescriptor[0]:  // Bytes 0-3
        uint16 startBlock
        uint16 blockCount
    ExtentDescriptor[1]:  // Bytes 4-7
        uint16 startBlock
        uint16 blockCount
    ExtentDescriptor[2]:  // Bytes 8-11
        uint16 startBlock
        uint16 blockCount
}
```

Unused extents: Set both fields to 0.
Example: File uses blocks 100-109 and 200-249:

```
Extent[0]: startBlock=100, blockCount=10
Extent[1]: startBlock=200, blockCount=50
Extent[2]: startBlock=0,   blockCount=0   // Unused
```

3.2.4 Alternate MDB - Critical Backup

The alternate MDB is an **exact copy** of the primary MDB.

Location Calculation

Precise formula:

$$alt\_offset = device\_size\_in\_bytes - 1024 \tag{3.2}$$

Example for 10 MB volume:

```
Device size: 10,485,760 bytes
Alt MDB at:  10,485,760 - 1,024 = 10,484,736 bytes
```

Verification:

```
FILESIZE=$(stat -c%s volume.hfs)
ALTOFFSET=$((FILESIZE - 1024))
xxd -s $ALTOFFSET -l 2 -p volume.hfs
# Expected: 4244 (same signature as primary)
```

Common mistake: Using `(num_sectors - 2) * 512` assumes 512-byte sectors. The specification is **always** "1024 bytes before end", regardless of sector size.

3.3 HFS B-Trees - Complete Specification

3.3.1 B-Tree Overview

HFS uses B-trees for the catalog (files/folders) and extents overflow (fragmented files).

Key Characteristics

- **Node size**: Fixed 512 bytes for HFS
- **Balancing**: Self-balancing tree structure
- **Key comparison**: Case-insensitive for catalog names
- **Depth**: Typically 2-4 levels for most volumes

3.3.2 Node Descriptor - First 14 Bytes of Every Node

Every B-tree node starts with this 14-byte header:

Offset	Field	Description
+0	fLink	Forward link: next node at this level (uint32)
+4	bLink	Backward link: previous node (uint32)
+8	kind	Node type (int8, see below)
+9	height	Node level: 0 = leaf, 1+ = index (uint8)
+10	numRecords	Number of records in node (uint16)
+12	reserved	Reserved, must be 0 (uint16)

Table 3.6: Node Descriptor (14 bytes)

Node Types (kind field)

Value	Meaning
-1 (0xFF)	Index node (internal)
0	Header node (node 0only)
1	Map node (allocation bitmap)
2	Leaf node (data records)

Table 3.7: B-Tree Node Types

Oddity: Index nodes use -1 (signed), not 255 (unsigned). This is a signed int8 field.

3.3.3 Header Node (Node 0) - Complete Structure

The first node (offset 0 in B-tree file) is always the header node.

BTHeaderRec Structure (106 bytes)

Located at offset 14 (after node descriptor):

Offset	Field	Type	Description
+14	treeDepth	uint16	Current depth (0 = empty)
+16	rootNode	uint32	Node number of root
+20	leafRecords	uint32	Total leaf records
+24	firstLeafNode	uint32	First leaf node number
+28	lastLeafNode	uint32	Last leaf node number
+32	nodeSize	uint16	Node size (**512 for HFS**)
+34	maxKeyLength	uint16	Maximum key length
+36	totalNodes	uint32	Total nodes in tree
+40	freeNodes	uint32	Unused nodes
+44	reserved1	uint16	Reserved
+46	clumpSize	uint32	Clump size (bytes)
+50	btreeType	uint8	0=Catalog, 255=Extents
+51	reserved2	uint8	Reserved
+52	attributes	uint32	B-tree attributes
+56	reserved3	uint8[64]	Reserved

Table 3.8: BTHeaderRec Structure

Critical: nodeSize MUST be 512 for HFS. HFS+ uses 4096, but HFS is fixed at 512.

3.3.4 Date/Time Representation

HFS uses Mac absolute time: unsigned 32-bit seconds since midnight, January 1, 1904 GMT.

Conversion formulas:

```
MAC_EPOCH = 2082844800  // Offset from Unix epoch

Unix to Mac: mac_time = unix_time + MAC_EPOCH
Mac to Unix: unix_time = mac_time - MAC_EPOCH
```

Y2K28 Problem:

$$Maxdate = 1904 + \frac{2^{32}}{365.25 \times 24 \times 3600} \approx 2028 \tag{3.3}$$

Specifically: February 6, 2028, 06:28:15 GMT

Safe date handling: hfsutils uses `hfs_get_safe_time()` which caps dates at 2028.

3.4 Byte Order (Endianness) - Critical

All HFS multi-byte fields are big-endian.

3.4.1 Endianness Examples

16-bit value 0x1234:

```
Byte 0: 0x12 (MSB)
Byte 1: 0x34 (LSB)
```

32-bit value 0x12345678:

```
Byte 0: 0x12 (Most significant)
Byte 1: 0x34
Byte 2: 0x56
Byte 3: 0x78 (Least significant)
```

Writing code:

```
// WRONG - host byte order
uint16_t value = 0x4244;
write(fd, &value, 2);  // Writes 0x44 0x42 on little-endian!

// CORRECT - explicit byte order
unsigned char sig[2] = {0x42, 0x44};
write(fd, sig, 2);     // Always correct
```

3.5 Oddities, Edge Cases, and Implementation Notes

3.5.1 The 16-Bit Limitation

HFS uses 16-bit allocation block numbers, limiting volumes to 65,535 blocks maximum.

Volume size calculation:

$$max\_volume = 65535 \times block\_size \tag{3.4}$$

For 32 KB blocks:

$$65535 \times 32768 = 2,147,450,880 bytes \approx 2GB \tag{3.5}$$

Limitation: Cannot create HFS volumes larger than ˜2 TB (with 64 KB blocks).

3.5.2 Pascal Strings vs C Strings

Pascal string (HFS): Length byte + data

```
"\x06Hello!"  // Byte 0 = length, followed by data
```

C string: Null-terminated

```
"Hello!\0"    // Ends with null byte
```

Gotcha: A Pascal string of length 0 is valid (empty string). A C string must have at least the null terminator.

3.5.3 Allocation Block Alignment

File data MUST start on allocation block boundaries. You cannot start a file in the middle of a block.

Implication: Small files waste space. If block size is 4 KB, a 1-byte file wastes 4095 bytes.

3.5.4 MacRoman Character Encoding

HFS uses MacRoman, not ASCII or UTF-8.

Differences from ASCII:

- Characters 0-127: Same as ASCII
- Characters 128-255: Different from ISO-8859-1

Example oddities:

- 0xD0 = en dash (—) in MacRoman, in ISO-8859-1
- 0xD1 = em dash (—) in MacRoman, Ñ in ISO-8859-1

Implementation: For maximum compatibility, restrict filenames to ASCII 32-126.

Chapter 4

HFS+ Specification

4.1 HFS+ Filesystem Overview

HFS Plus (HFS+), also known as Mac OS Extended, is Apple's modern filesystem designed to replace HFS. Introduced in Mac OS 8.1 (1998), it addresses HFS limitations while maintaining backward compatibility.

4.1.1 Key Improvements over HFS

- **Unicode filenames**: Full Unicode support (up to 255 UTF-16 characters)
- **Larger volumes**: Up to 8 EB (exabytes) theoretical
- **Larger files**: Up to 8 EB per file
- **Smaller allocation blocks**: More efficient space usage
- **Journaling support**: Optional transaction journal for crash recovery
- **Hard links**: Support for hard links (Mac OS X 10.2+)
- **Symbolic links**: Support for symbolic links
- **Extended attributes**: Arbitrary metadata on files/folders
- **Case sensitivity option**: HFSX variant supports case-sensitive names

4.1.2 HFS+ Characteristics

Feature	**Specification**
Maximum volume size	8 EB (2^{63} bytes)
Maximum file size	8 EB (2^{63} bytes)
Filename length	255 Unicode characters (UTF-16)
Date range	January 1, 1904 to February 6, 2040 (Y2K40)
Minimum allocation block	512 bytes
Block addressing	32-bit allocation block numbers
Case sensitivity	Case-insensitive (HFS+) or case-sensitive (HFSX)

Table 4.1: HFS+ Specifications

4.2 Volume Structure

HFS+ volumes are structured similarly to HFS but with enhanced metadata structures.

4.2.1 Volume Layout

Location	Name	Description
Byte 0	Reserved	Boot sector (512 bytes)
Byte 512	Reserved	Additional boot area (512 bytes)
Byte 1024	Volume Header	Primary volume metadata
...	Allocation Bitmap	Volume space allocation map
...	Allocation File	B-tree of allocation bitmap extents
...	Extents Overflow File	B-tree of file extent records
...	Catalog File	B-tree of all files and folders
...	Attributes File	B-tree of extended attributes
...	Startup File	Boot loader for non-Mac systems
...	Data Area	File contents and special files
-1024	Alternate VH	Backup copy of Volume Header

Table 4.2: HFS+ Volume Layout

4.3 Volume Header - Complete Bit-Level Specification

The Volume Header is the **most critical structure** in HFS+. Located at byte offset 1024, it is exactly **512 bytes**.

4.3.1 Volume Header Complete Field Map - Every Byte Documented

Offset	Field	Type	Size	Description
+0	signature	uint16	2	**0x482B** ('H+') or **0x4858** ('HX')
+2	version	uint16	2	**4** (HFS+) or **5** (HFSX)
+4	attributes	uint32	4	Volume attributes (flags, see below)
+8	lastMountedVersion	uint32	4	OS signature that last mounted
+12	journalInfoBlock	uint32	4	Journal info block (0 = no journal)
+16	createDate	uint32	4	Creation date (HFS+ time)
+20	modifyDate	uint32	4	Last modification date
+24	backupDate	uint32	4	Last backup date
+28	checkedDate	uint32	4	Last fsck date
+32	fileCount	uint32	4	Total files on volume
+36	folderCount	uint32	4	Total folders on volume
+40	blockSize	uint32	4	Allocation block size (bytes)
+44	totalBlocks	uint32	4	Total allocation blocks
+48	freeBlocks	uint32	4	Free allocation blocks
+52	nextAllocation	uint32	4	Hint for next allocation

Offset	Field	Type	Size	Description
+56	rsrcClumpSize	uint32	4	Default resource fork clump
+60	dataClumpSize	uint32	4	Default data fork clump
+64	nextCatalogID	uint32	4	Next unused Catalog Node ID
+68	writeCount	uint32	4	Volume write count
+72	encodingsBitmap	uint64	8	Text encodings used (64 bits)
+80	finderInfo	uint32[8]	32	Finder information
+112	allocationFile	HFSPlusForkData	80	Allocation file fork
+192	extentsFile	HFSPlusForkData	80	Extents file fork
+272	catalogFile	HFSPlusForkData	80	Catalog file fork
+352	attributesFile	HFSPlusForkData	80	Attributes file fork
+432	startupFile	HFSPlusForkData	80	Startup file fork

Table 4.3: HFS+ Volume Header Structure (512 bytes total)

Total size verification: 2 + 2 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 8 + 32 + (80*5) = 512 bytes

4.3.2 Critical Field Details - Bit-by-Bit

signature - Volume Signature (Offset +0, 2 bytes)

HFS+ Standard: `0x482B` (big-endian)
Byte representation:

```
Offset 1024: 0x48 ('H')
Offset 1025: 0x2B ('+')
```

HFSX Case-Sensitive: `0x4858` (big-endian)
Byte representation:

```
Offset 1024: 0x48 ('H')
Offset 1025: 0x58 ('X')
```

Validation:

- Must be exactly `0x482B` or `0x4858`
- Any other value = not HFS+ or corrupted
- `0x4244` = HFS (not HFS+)

Hex dump verification:

```
xxd -s 1024 -l 2 -p volume.hfsplus
Expected: 482b (HFS+) or 4858 (HFSX)
```

version - Volume Version (Offset +2, 2 bytes)

HFS+ Standard: `0x0004` (4 decimal, big-endian)
Byte representation:

```
Offset 1026: 0x00
Offset 1027: 0x04
```

HFSX: `0x0005` (5 decimal)
Validation: mkfs.hfs+ sets version to 4 for standard HFS+.

attributes - Volume Attributes (Offset +4, 4 bytes)

32-bit flags field (big-endian). **Every bit documented**:

Bit	Hex Mask	Meaning
0-6	0x0000007F	Reserved (must be 0)
7	0x00000080	Volume locked by hardware
8	**0x00000100**	**Volume unmounted cleanly**
9	0x00000200	Volume has spared bad blocks
10	0x00000400	Needs fsck (consistency check required)
11	0x00000800	Catalog node IDs wrapped around
12	0x00001000	Software lock
13	**0x00002000**	**Volume is journaled**
14	0x00004000	Reserved
15	0x00008000	Reserved
16-31	0xFFFF0000	Reserved

Table 4.4: Volume Header attributes Bit Definitions

Common values:

- `0x00000100`: Clean, non-journaled (mkfs.hfs+ default)
- `0x00002100`: Clean, journaled (mkfs.hfs+ -j)

Byte representation for 0x00000100:

```
Offset 1028: 0x00
Offset 1029: 0x00
Offset 1030: 0x01
Offset 1031: 0x00
```

Verification:

```
xxd -s 1028 -l 4 -p volume.hfsplus
Expected: 00000100 (non-journaled) or 00002100 (journaled)
```

blockSize - Allocation Block Size (Offset +40, 4 bytes)

Value: 32-bit unsigned, big-endian
Valid range:

- Minimum: 512 bytes
- Maximum: Typically 32 KB, theoretically larger
- Must be power of 2
- Must be multiple of 512

Example for 4096 bytes (4 KB):

```
Decimal: 4096
Hex: 0x00001000
Bytes at offset 1064:
  0x00 0x00 0x10 0x00
```

Validation:

```
xxd -s 1064 -l 4 -p volume.hfsplus
# For 4 KB blocks: 00001000
# For 8 KB blocks: 00002000
```

rsrcClumpSize and dataClumpSize (Offsets +56, +60)

Critical: These MUST be non-zero in valid HFS+ volumes.

Recommended value:

$$clumpSize = blockSize \times 4 \tag{4.1}$$

For 4 KB blocks:

$$clumpSize = 4096 \times 4 = 16384 bytes = \texttt{0x00004000} \tag{4.2}$$

Byte representation:

```
rsrcClumpSize at offset 1080: 0x00 0x00 0x40 0x00
dataClumpSize at offset 1084: 0x00 0x00 0x40 0x00
```

Common error: If these are 0x00000000, the Volume Header is invalid and nextCatalogID appears at wrong offset.

Verification:

```
xxd -s 1080 -l 4 -p volume.hfsplus  # rsrcClumpSize
xxd -s 1084 -l 4 -p volume.hfsplus  # dataClumpSize
# Both should be non-zero
```

nextCatalogID - Next CNID (Offset +64, 4 bytes)

Minimum: `0x00000010` (16 decimal)

Reserved CNIDs 1-15:

CNID	Purpose
1	Root folder's parent (kHFSRootParentID)
2	Root folder (kHFSRootFolderID)
3	Extents overflow file (kHFSExtentsFileID)
4	Catalog file (kHFSCatalogFileID)
5	Bad allocation blocks file (kHFSBadBlockFileID)
6	Allocation bitmap file (kHFSAllocationFileID)
7	Startup file (kHFSStartupFileID)
8	Attributes file (kHFSAttributesFileID)
9-13	Reserved
14	Journal file (kHFSJournalFileID, if journaled)
15	Journal info block (kHFSJournalInfoBlockID)

Table 4.5: Reserved HFS+ Catalog Node IDs

Byte representation for value 16:

```
Offset 1088: 0x00
Offset 1089: 0x00
Offset 1090: 0x00
Offset 1091: 0x10
```

Critical validation:

```
xxd -s 1088 -l 4 -p volume.hfsplus
Expected: 00000010 (minimum value)
```

Common error: If you see `00000000`, rsrcClumpSize/dataClumpSize were likely omitted, shifting all subsequent fields.

4.3.3 HFSPlusForkData Structure - 80 Bytes Per Fork

Each fork descriptor is 80 bytes:

Offset	Field	Description
+0	logicalSize	File size in bytes (uint64)
+8	clumpSize	Clump size for this file (uint32)
+12	totalBlocks	Total allocation blocks (uint32)
+16	extents[0]	First extent descriptor (8 bytes)
+24	extents[1]	Second extent descriptor (8 bytes)
+32	extents[2]	Third extent descriptor (8 bytes)
+40	extents[3]	Fourth extent descriptor (8 bytes)
+48	extents[4]	Fifth extent descriptor (8 bytes)
+56	extents[5]	Sixth extent descriptor (8 bytes)
+64	extents[6]	Seventh extent descriptor (8 bytes)
+72	extents[7]	Eighth extent descriptor (8 bytes)

Table 4.6: HFSPlusForkData Structure (80 bytes)

HFSPlusExtentDescriptor - 8 Bytes Each

Offset	Field	Description
+0	startBlock	First allocation block (uint32)
+4	blockCount	Number of blocks (uint32)

Table 4.7: HFSPlusExtentDescriptor (8 bytes)

Example: Catalog file uses blocks 100-199:

```
logicalSize:  0x0000000000018800  (100 KB)
clumpSize:    0x00004000          (16 KB)
totalBlocks:  0x00000019          (25 blocks)
extents[0]:   startBlock=100, blockCount=25
              Bytes: 00 00 00 64 00 00 00 19
extents[1-7]: startBlock=0, blockCount=0 (unused)
```

4.3.4 Alternate Volume Header Location

Exact formula:

$$\text{alt\_VH\_offset} = \text{volume\_size\_bytes} - 1024 \tag{4.3}$$

Same as HFS. This is **NOT** sector-based.

Example for 50 MB:

```
Volume size: 52,428,800 bytes
Alt VH at:   52,428,800 - 1,024 = 52,427,776 bytes
```

Verification:

```
FILESIZE=$(stat -c%s volume.hfsplus)
ALTOFFSET=$((FILESIZE - 1024))
xxd -s $ALTOFFSET -l 2 -p volume.hfsplus
Expected: 482b (same as primary)
```

4.4 HFS+ B-Trees - Complete Node and Record Structures

HFS+ uses B-trees for **all metadata organization**: Catalog, Extents, Attributes.

4.4.1 B-Tree Node Structure - 14-Byte Node Descriptor

Every B-tree node begins with a 14-byte descriptor:

Offset	Field	Type	Size	Description
+0	fLink	uint32	4	Forward link (next node, 0 = none)
+4	bLink	uint32	4	Backward link (prev node, 0 = none)
+8	kind	int8	1	Node type (see below)
+9	height	uint8	1	Level in tree (0 = leaf)
+10	numRecords	uint16	2	Number of records in node
+12	reserved	uint16	2	Reserved (must be 0)

Table 4.8: BTNodeDescriptor (14 bytes)

kind - Node Type Values

Value	Node Type
-1	Leaf node (contains data records)
0	Index node (contains child pointers)
1	Header node (B-tree metadata, always node 0)
2	Map node (allocation bitmap)

Table 4.9: B-tree Node Types

Byte example for header node (kind=1):

```
Offset +8: 0x01 (header node)
Offset +9: 0x00 (height 0, not used in header)
Offset +10-11: 0x00 0x03 (3 records typical: header, user data, map)
```

4.4.2 BTHeaderRec - B-Tree Header Record (106 Bytes)

Located in the **first record of node 0 (header node)** in every B-tree.

Offset	Field	Type	Size	Description
+0	treeDepth	uint16	2	Current depth (1 = only root)
+2	rootNode	uint32	4	Root node number
+6	leafRecords	uint32	4	Total leaf records
+10	firstLeafNode	uint32	4	First leaf node number
+14	lastLeafNode	uint32	4	Last leaf node number
+18	nodeSize	uint16	2	Node size in bytes
+20	maxKeyLength	uint16	2	Maximum key length
+22	totalNodes	uint32	4	Total nodes allocated
+26	freeNodes	uint32	4	Number of free nodes
+30	reserved1	uint16	2	Reserved
+32	clumpSize	uint32	4	Clump size for B-tree file
+36	btreeType	uint8	1	B-tree type (0=HFS, 128=HFS+)
+37	keyCompareType	uint8	1	Key comparison type
+38	attributes	uint32	4	B-tree attributes (flags)
+42	reserved3	uint32[16]	64	Reserved (must be 0)

Table 4.10: BTHeaderRec Structure (106 bytes total)

Total size verification: 2 + 4 + 4 + 4 + 4 + 2 + 2 + 4 + 4 + 2 + 4 + 1 + 1 + 4 + 64 = 106 bytes

Critical BTHeaderRec Field Details

treeDepth (Offset +0):

- Value 0: Empty tree
- Value 1: Only root node (contains data directly)
- Value 2+: Root is index node
- **New volume**: Usually 1 (minimal tree)

nodeSize (Offset +18):

- **HFS+ Standard**: 4096 bytes (4 KB)
- **HFS+ Large**: 8192 bytes (8 KB)
- Must be power of 2
- Must be $\geq$ 512 bytes

Byte representation for 4096:

```
Offset +18: 0x10
Offset +19: 0x00
(Big-endian: 0x1000 = 4096)
```

btreeType (Offset +36):

- **0**: HFS B-tree (legacy)
- **128**: HFS+ B-tree (standard)

- **255**: Reserved

keyCompareType (Offset +37):

- **0xBC**: Case-insensitive (HFS+ default)
- **0xCF**: Binary compare (HFSX case-sensitive)

attributes (Offset +38, 4 bytes):

Bit	Hex Mask	Meaning
0	0x00000001	Bad close (B-tree not closed properly)
1	0x00000002	Big keys (key length > 255 bytes)
2	0x00000004	Variable index keys
3-31	0xFFFFFFF8	Reserved (must be 0)

Table 4.11: BTHeaderRec attributes Flags

4.4.3 Catalog File B-Tree - Complete Key and Record Formats

The Catalog File contains **all files and folders**. It uses **Unicode HFSUniStr255 keys**.

HFSPlusCatalogKey - Variable Length

Offset	Field	Type	Size	Description
+0	keyLength	uint16	2	Total key length (excluding this field)
+2	parentID	uint32	4	Parent folder CNID
+6	nodeName	HFSUniStr255	var	Unicode filename (see below)

Table 4.12: HFSPlusCatalogKey Structure

HFSUniStr255 - Unicode String (Max 255 UTF-16 Characters)

Offset	Field	Description
+0	length	uint16: Number of UTF-16 characters (NOT bytes)
+2	unicode	uint16[]: UTF-16BE characters

Table 4.13: HFSUniStr255 Structure

Example: Filename ”Test.txt” (8 characters)

```
length:  0x0008 (8 UTF-16 chars)
unicode:
  0x0054 ('T')
  0x0065 ('e')
  0x0073 ('s')
  0x0074 ('t')
  0x002E ('.')
```

```
  0x0074 ('t')
  0x0078 ('x')
  0x0074 ('t')
Total: 2 + (8*2) = 18 bytes for nodeName
```

Complete catalog key for "Test.txt" in root (CNID 2):

```
keyLength: 0x0016 (22 bytes: 4 parentID + 18 nodeName)
parentID:  0x00000002 (root folder)
nodeName:  [18 bytes as shown above]
Total key: 2 + 22 = 24 bytes
```

Catalog Record Types - 4 Types

Type	Value	Description
kHFSPlusFolderRecord	0x0001	Folder (directory)
kHFSPlusFileRecord	0x0002	File
kHFSPlusFolderThreadRecord	0x0003	Folder thread (CNID → name)
kHFSPlusFileThreadRecord	0x0004	File thread (CNID → name)

Table 4.14: Catalog Record Type Values

HFSPlusCatalogFile - File Record (248 Bytes)

Offset	Field	Type	Size	Description
+0	recordType	int16	2	**0x0002** (file)
+2	flags	uint16	2	File flags
+4	reserved1	uint32	4	Reserved
+8	fileID	uint32	4	Catalog Node ID (CNID)
+12	createDate	uint32	4	Creation date (HFS+ time)
+16	contentModDate	uint32	4	Content modification date
+20	attributeModDate	uint32	4	Attribute modification date
+24	accessDate	uint32	4	Last access date
+28	backupDate	uint32	4	Backup date
+32	permissions	HFSPlusBSDInfo	16	BSD ownership/permissions
+48	userInfo	FInfo	16	Finder user info
+64	finderInfo	FXInfo	16	Finder extended info
+80	textEncoding	uint32	4	Text encoding hint
+84	reserved2	uint32	4	Reserved
+88	dataFork	HFSPlusForkData	80	Data fork descriptor
+168	resourceFork	HFSPlusForkData	80	Resource fork descriptor

Table 4.15: HFSPlusCatalogFile Structure (248 bytes total)

Total size verification: $2 + 2 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 16 + 16 + 16 + 4 + 4 + 80 + 80 = 248$ bytes

4.4.4 Catalog File

The Catalog File is a B-tree containing all files and folders on the volume.

Catalog Keys

- **keyLength**: 2 bytes
- **parentID**: 4 bytes (Parent folder CNID)
- **nodeName**: Variable-length Unicode string

Catalog Record Types

1. **Folder Record (0x0001)**: Directory metadata
2. **File Record (0x0002)**: File metadata and fork information
3. **Folder Thread (0x0003)**: Maps CNID to parent folder
4. **File Thread (0x0004)**: Maps file CNID to parent folder

File Record Structure

- recordType: 0x0002
- flags: File flags
- fileID: Catalog Node ID
- createDate, modifyDate: Timestamps
- permissions: Unix permissions
- userInfo, finderInfo: Finder metadata
- textEncoding: Filename encoding hint
- dataFork: Fork data for data fork (80 bytes)
- resourceFork: Fork data for resource fork (80 bytes)

4.4.5 Extents Overflow File

B-tree storing additional extent records when a file's fork exceeds the 8 extents stored in the catalog.

Extent Key

- keyLength: 2 bytes
- forkType: 0x00 (data) or 0xFF (resource)
- fileID: Catalog Node ID
- startBlock: Starting allocation block

Extent Descriptor

- startBlock: Starting allocation block (4 bytes)
- blockCount: Number of contiguous blocks (4 bytes)

4.4.6 Attributes File

B-tree storing extended attributes (metadata) for files and folders.

Supported Attributes

- Extended attributes (xattrs)
- Access Control Lists (ACLs)
- Resource fork data (if not inline)
- Compressed data (HFS+ compression)

1. fsck.hfs+ detects journal
2. Scans journal for uncommitted transactions
3. Replays completed transactions to restore consistency
4. Marks volume clean after successful replay

4.5 Unicode Normalization - CRITICAL for Filename Compatibility

HFS+ uses **Unicode Normalization Form D (NFD)** for all filenames. This is **mandatory** and causes significant compatibility issues with other systems.

4.5.1 NFD vs NFC - The Core Problem

Unicode allows multiple representations of the same character:

- **NFC (Composed)**: Single codepoint for accented characters
- **NFD (Decomposed)**: Base character + combining accent

Example: Letter ”é” (e with acute accent)

Form	Representation
NFC	U+00E9 (single codepoint: LATIN SMALL LETTER E WITH ACUTE)
NFD	U+0065 U+0301 (two codepoints: LATIN SMALL LETTER E + COMBINING ACUTE ACCENT)

Table 4.16: Unicode Normalization Example

Byte representation in UTF-16BE:

```
NFC (1 UTF-16 unit):  0x00E9
NFD (2 UTF-16 units): 0x0065 0x0301

In HFSUniStr255:
  length (NFC): 0x0001 (1 character)
  length (NFD): 0x0002 (2 characters)
```

4.5.2 HFS+ NFD Requirement - MANDATORY

Apple Technical Note TN1150: All HFS+ filenames MUST be stored in NFD form.
Conversion algorithm:

1. Receive filename from user (may be in any form)
2. Decompose to NFD using Unicode decomposition tables
3. Store in catalog with NFD form
4. When reading, return NFD form to user

Critical implementation detail:

- mkfs.hfs+ must accept filenames and convert to NFD
- Catalog B-tree keys are compared in NFD form
- Case-insensitive comparison uses Unicode case folding tables

4.5.3 Common NFD Characters - Complete Table

Character	NFC	NFD	Description
à	U+00E0	U+0061 U+0300	a + grave
á	U+00E1	U+0061 U+0301	a + acute
â	U+00E2	U+0061 U+0302	a + circumflex
ã	U+00E3	U+0061 U+0303	a + tilde
ä	U+00E4	U+0061 U+0308	a + diaeresis
ñ	U+00F1	U+006E U+0303	n + tilde
ç	U+00E7	U+0063 U+0327	c + cedilla
ü	U+00FC	U+0075 U+0308	u + diaeresis
ö	U+00F6	U+006F U+0308	o + diaeresis
å	U+00E5	U+0061 U+030A	a + ring above

Table 4.17: Common NFD Decompositions

4.5.4 Compatibility Issues

Linux/Windows: Use NFC by default
Problem: Filename created on macOS with "café.txt" (NFD) appears as different file than "café.txt" (NFC) created on Linux on the same HFS+ volume.
Workaround: Always normalize to NFD when writing to HFS+.
Implementation in hfsutils:

```
// Pseudo-code for filename conversion
void normalize_to_nfd(uint16_t *unicode, size_t *length) {
    // For each character:
    //   1. Look up in Unicode decomposition table
    //   2. Replace with base + combining characters
    //   3. Update length accordingly
}
```

4.6 HFS+ Time Format - Mac Epoch and Conversion

HFS+ uses a **32-bit unsigned integer** for all timestamps, representing seconds since the **Mac epoch**.

4.6.1 Mac Epoch Definition

Mac epoch: January 1, 1904 00:00:00 UTC
Unix epoch: January 1, 1970 00:00:00 UTC
Difference: 2,082,844,800 seconds (66 years)

4.6.2 Date Range

With 32-bit unsigned integer:

- Minimum: 0 (January 1, 1904)
- Maximum: 4,294,967,295 (February 6, 2040 06:28:15 UTC)

Y2K40 Problem: HFS+ timestamps overflow on February 6, 2040.

4.6.3 Conversion Formulas

HFS+ to Unix time:

$$\text{unix\_time} = \text{hfs\_time} - 2082844800 \tag{4.4}$$

Unix to HFS+ time:

$$\text{hfs\_time} = \text{unix\_time} + 2082844800 \tag{4.5}$$

Example conversion:

```
HFS+ time:  3600000000 (0xD693A400)
Unix time:  3600000000 - 2082844800 = 1517155200
Unix date:  January 28, 2018 16:00:00 UTC
```

4.6.4 Byte Representation

All timestamps are big-endian 32-bit unsigned integers.
Example: December 25, 2020 12:00:00 UTC

```
Unix timestamp: 1608897600
HFS+ timestamp: 1608897600 + 2082844800 = 3691742400
Hex: 0xDBF49140
Bytes: 0xDB 0xF4 0x91 0x40
```

Verification in Volume Header createDate (offset +16):

```
xxd -s 1040 -l 4 -p volume.hfsplus
Expected format: DBXXXXXX (for recent dates)
```

4.6.5 Y2K40 Safeguards in hfsutils

Implementation in src/common/hfstime.c:

```
#define HFS_Y2K40_LIMIT 4294967295
#define HFS_SAFE_YEAR_2030 4102444800

uint32_t hfs_get_safe_time(void) {
    time_t now = time(NULL);
    uint32_t hfs_time = (uint32_t)now + 2082844800;

    // If beyond Y2K40, use January 1, 2030
    if (hfs_time > HFS_Y2K40_LIMIT) {
        hfs_time = HFS_SAFE_YEAR_2030;
    }

    return hfs_time;
}
```

Critical: mkfs.hfs, mkfs.hfs+, and fsck.hfs+ all use this function.

4.7 HFS+ Critical Oddities and Edge Cases

4.7.1 Case-Insensitive vs Case-Preserving

HFS+ Standard Behavior:

- **Case-preserving**: Stores "MyFile.txt" as typed
- **Case-insensitive**: "myfile.txt" and "MyFile.txt" are the SAME file
- Uses Unicode case folding for comparison

HFSX Behavior:

- **Case-sensitive**: "myfile.txt" and "MyFile.txt" are DIFFERENT files
- Signature: 0x4858 ('HX'), version 5
- keyCompareType: 0xCF (binary compare)

Incompatibility: Standard HFS+ cannot be converted to HFSX without reformatting.

4.7.2 Folder Valence - Hidden Complexity

HFSPlusCatalogFolder structure includes "valence" field:

- Counts number of items in folder
- **Does NOT include invisible files** (e.g., .DS_Store)
- Must be updated on every file creation/deletion
- Inconsistency causes fsck errors

4.7.3 Hard Links - Indirect Nodes

HFS+ supports hard links (multiple names for same file):

- Uses **indirect nodes** with special parent ID
- Hard link parent: 0xFFFFFFFE (reserved)
- Each hard link has unique CNID
- All point to same fileID in hidden directory

Implementation complexity: Requires special catalog traversal logic.

4.7.4 Compression - Undocumented Extension

macOS 10.6+ introduced HFS+ compression (unofficial):

- Compressed data stored in **extended attributes**
- Resource fork contains decompression metadata
- **Not part of original HFS+ spec**
- Third-party implementations typically ignore

4.7.5 Journal Checksum Algorithm - Missing from TN1150

Journal uses CRC32 or similar checksum (not fully documented):

- Checksum in journal header (offset +28)
- Verifies journal integrity before replay
- **Algorithm varies by implementation**

Safe approach: If checksum fails, refuse to replay journal (mount read-only).

4.7.6 Allocation Block Alignment

Critical for performance:

- Allocation blocks should align to physical sectors
- blockSize should be multiple of physical sector size
- Modern drives: 4 KB sectors → use 4 KB allocation blocks
- Misalignment causes read-modify-write penalty

mkfs.hfs+ default: 4096 bytes (optimal for modern drives)

4.7.7 Extended Attributes File - Optional

attributesFile in Volume Header (offset +352):

- Can be empty (logicalSize = 0) on new volumes
- Created on-demand when first extended attribute added
- Uses its own B-tree structure
- Keys: (fileID, attribute name)

Common attributes:

- com.apple.FinderInfo: Finder metadata
- com.apple.ResourceFork: Resource fork data (alternative storage)
- com.apple.decmpfs: Compressed file data

4.8 Reimplementation Checklist - Everything You Need

4.8.1 Data Structures Required

1. Volume Header (512 bytes) - Complete in this document
2. HFSPlusForkData (80 bytes) - Complete in this document
3. HFSPlusExtentDescriptor (8 bytes) - Complete in this document
4. BTNodeDescriptor (14 bytes) - Complete in this document
5. BTHeaderRec (106 bytes) - Complete in this document
6. HFSPlusCatalogKey (variable) - Complete in this document
7. HFSUniStr255 (variable, max 512 bytes) - Complete in this document
8. HFSPlusCatalogFile (248 bytes) - Complete in this document
9. HFSPlusCatalogFolder (88 bytes) - See Apple TN1150
10. JournalInfoBlock (96 bytes) - Complete in this document

4.8.2 Algorithms Required

1. Unicode NFD normalization (use ICU library or tables)
2. Unicode case folding (for case-insensitive comparison)
3. B-tree insertion/deletion (standard CS algorithm)
4. Extent allocation/deallocation
5. Bitmap manipulation (allocation file)
6. CRC32 or checksum (for journal)
7. HFS+ time conversion (formulas in this document)

4.8.3 Validation Commands

All xxd commands in this document can verify:

- Volume signature (offset 1024)
- Volume version (offset 1026)
- Attributes flags (offset 1028)
- blockSize (offset 1064)
- rsrcClumpSize, dataClumpSize (offsets 1080, 1084)
- nextCatalogID (offset 1088)
- Alternate Volume Header (volume_size - 1024)

fsck.hfs+ validates:

- All B-tree structures
- Folder valence consistency
- Allocation bitmap consistency
- Extent overflow records
- Journal integrity (if present)

4.8.4 No External References Needed

This document contains:

- Every byte offset for critical structures
- All bit flags with hex masks
- Complete formulas for calculations
- Byte examples for verification
- Common error patterns
- Compatibility warnings

You can reimplement HFS+ with:

1. This chapter (complete specification)
2. Standard Unicode tables (NFD decomposition)
3. Standard B-tree algorithm (CS textbook)
4. CRC32 implementation (standard)

No internet required after you have these resources.

4.9 HFS+ vs HFSX

HFSX is a variant of HFS+ with case-sensitive filename comparison.

Note: hfsutils mkfs.hfs+ creates standard HFS+ volumes (case-insensitive). HFSX support is not currently implemented.

Feature	**HFS+**	**HFSX**
Signature	0x482B ('H+')	0x4858 ('HX')
Version	4	5
Case sensitivity	No	Yes
Filename comparison	Case-insensitive	Case-sensitive
"File.txt" = "file.txt"	Yes	No

Table 4.18: HFS+ vs HFSX

4.10 Compatibility Considerations

4.10.1 macOS

- Native support for HFS+ and HFSX
- Full journaling support
- All extended attributes supported
- APFS is now default (macOS 10.13+)

4.10.2 Linux

- Kernel module `hfsplus` required
- **No journaling support in kernel driver**
- Basic read/write support
- Some extended attributes supported
- May mount journaled volumes read-only for safety

4.10.3 FreeBSD/OpenBSD/NetBSD

- Native HFS+ read support
- Write support via FUSE
- No journaling support

4.10.4 Windows

- No native HFS+ support
- Third-party drivers available (Paragon, MacDrive)
- Boot Camp drivers provide read-only access

Chapter 5

mkfs Utilities

5.1 mkfs.hfs - HFS Filesystem Creation

This chapter will include complete mkfs.hfs and mkfs.hfs+ documentation from manpages.

5.1.1 Command Synopsis

```
mkfs.hfs [-f] [-L label] device
mkfs.hfs+ [-f] [-j] [-L label] [-s size] device
```

Note: Full content will be converted from doc/man/mkfs.hfs.8 and mkfs.hfplus.8

Chapter 6

fsck Utilities

6.1 fsck.hfs - HFS Filesystem Check

This chapter will include complete fsck.hfs and fsck.hfs+ documentation from manpages.

6.1.1 Command Synopsis

```
fsck.hfs [-dfnpvy] device
fsck.hfs+ [-dfnpvy] device
```

Note: Full content will be converted from doc/man/fsck.hfs.8 and fsck.hfs+.8

Chapter 7

mount Utilities

7.1 mount.hfs - HFS Filesystem Mounting

This chapter will include complete mount.hfs and mount.hfs+ documentation from manpages.

7.1.1 Command Synopsis

```
mount.hfs [-o options] [-r] [-w] [-v] device mountpoint
mount.hfs+ [-o options] [-r] [-w] [-v] device mountpoint
```

Note: Full content will be converted from doc/man/mount.hfs.8 and mount.hfs+.8

Chapter 8

hfsutil Commands

8.1 hfsutil Commands

This chapter will include hfsutil command documentation.

8.1.1 Commands

- hformat - Format HFS volumes
- hmount - Mount HFS volumes
- humount - Unmount HFS volumes
- hls - List directory contents
- hcopy - Copy files
- hmkdir - Create directories
- And more...

Note: Full content will be converted from doc/man/hfsutils.1 and related manpages

Chapter 9

Implementation Details

9.1 Implementation Details

This chapter will include implementation details from HFS_IMPLEMENTATION_NOTES.md

9.1.1 Topics to Cover

- Code structure
- B-tree algorithms
- Validation strategies
- Repair mechanisms

Chapter 10

Testing and Validation

10.1 Testing and Validation

This chapter will include test suite documentation.

10.1.1 Zero-Tolerance Policy

All filesystems must be 100% correct. Any deviation from specification results in test failure.

10.1.2 Test Suite

- test/test_mkfs.sh - Filesystem creation tests
- test/test_fsck.sh - Validation tests
- test/test_hfsutils.sh - Command tests

Appendix A

Appendix

A.1 Structure Definitions

This appendix will include complete structure definitions for HFS and HFS+.

A.2 Error Codes

This section will list all error codes used by the utilities.

A.3 Glossary

Allocation Block Unit of disk space allocation

B-tree Balanced tree data structure

Catalog Directory of all files and folders

CNID Catalog Node ID

Extent Contiguous range of allocation blocks

Fork Part of a file (data or resource)

MDB Master Directory Block (HFS)

Volume Header Main metadata structure (HFS+)

A.4 References

- Inside Macintosh: Files
- Apple Technical Note TN1150
- Linux HFS/HFS+ kernel documentation
- BSD filesystem documentation