File Manager Reference

Data Management: File Management

2009-11-17

Apple Inc.
© 2001, 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AppleShare, AppleTalk, Carbon, Cocoa, ColorSync, Logic, Mac, Mac OS, Macintosh, ProDOS, Quartz, and QuickTime are trademarks of Apple Inc., registered in the United States and other countries.

Finder is a trademark of Apple Inc.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

File Manager Reference 15

Overview 15
Functions by Task 15
- Accessing Information About Files and Directories 15
- Accessing the Desktop Database 16
- Allocating Storage for Files 18
- Closing Files 19
- Comparing File System References 19
- Controlling Directory Access 19
- Controlling Login Access 20
- Converting Between Paths and FSRef Structures 20
- Copying and Moving Files 20
- Copying and Moving Objects Using Asynchronous High-Level File Operations 21
- Copying and Moving Objects Using Synchronous High-Level File Operations 21
- Creating a File System Reference (FSRef) 22
- Creating and Deleting File ID References 22
- Creating and Deleting Named Forks 23
- Creating Directories 23
- Creating File System Specifications 23
- Creating Files 24
- Creating, Calling, and Deleting Universal Procedure Pointers 24
- Deleting Files and Directories 25
- Determining the Unicode Names of the Data and Resource Forks 25
- Exchanging the Contents of Two Files 25
- Getting and Setting Volume Information 26
- Getting Volume Attributes 27
- Iterating Over Named Forks 27
- Locking and Unlocking File Ranges 27
- Locking and Unlocking Files and Directories 28
- Manipulating File and Fork Size 28
- Manipulating File Position 29
- Manipulating the Default Volume 29
- Mounting and Unmounting Volumes 30
- Mounting Remote Volumes 31
- Moving and Renaming Files or Directories 31
- Obtaining File and Directory Information Using a Catalog Iterator on HFS Plus Volumes 32
- Obtaining File Control Block Information 32
- Obtaining Fork Control Block Information 32
- Opening Files 32
- Opening Files While Denying Access 33
- Reading and Writing Files 34

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Resolving File ID References 34
Searching a Volume 34
Searching a Volume Using a Catalog Iterator 35
Updating Files 35
Updating Volumes 35
Using Change Notifications 36
Not Recommended 36
Functions 37
DisposeFNSubscriptionUPP 37
DisposeFSVolumeEjectUPP 38
DisposeFSVolumeMountUPP 38
DisposeFSVolumeUnmountUPP 38
DisposeIOCompletionUPP 39
FNGetDirectoryForSubscription 39
FNNotify 40
FNNotifyAll 40
FNNotifyByPath 41
FNSubscribe 41
FNSubscribeByPath 42
FNUnsubscribe 43
FSAllocateFork 43
FSCancelVolumeOperation 44
FSCatalogSearch 45
FSCloseFork 47
FSCloseIterator 48
FSCompareFSRefs 48
FSCopyDiskIDForVolume 49
FSCopyObjectAsync 50
FSCopyObjectSync 51
FSCopyURLForVolume 51
FSCreateDirectoryUnicode 52
FSCreateFileUnicode 54
FSCreateFork 55
FSCreateVolumeOperation 56
FSDeleteFork 56
FSDeleteObject 57
FSDisposeVolumeOperation 57
FSEjectVolumeAsync 58
FSEjectVolumeSync 59
FSExchangeObjects 59
FSFileOperationCancel 60
FSFileOperationCopyStatus 61
FSFileOperationCreate 61
FSFileOperationGetTypeID 62
FSFileOperationScheduleWithRunLoop 62
FSFileOperationUnscheduleFromRunLoop 63

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSFlushFork 64
FSFlushVolume 64
FSGetAsyncEjectStatus 65
FSGetAsyncMountStatus 65
FSGetAsyncUnmountStatus 66
FSGetCatalogInfo 67
FSGetCatalogInfoBulk 68
FSGetDataForkName 70
FSGetForkCBInfo 71
FSGetForkPosition 72
FSGetForkSize 73
FSGetResourceForkName 73
FSGetVolumeInfo 74
FSGetVolumeMountInfo 76
FSGetVolumeMountInfoSize 76
FSGetVolumeParms 76
FSIterateForks 77
FSLockRange 78
FSMakeFSRefUnicode 78
FSMountLocalVolumeAsync 79
FSMountLocalVolumeSync 80
FSMountServerVolumeAsync 80
FSMountServerVolumeSync 82
FSMoveObject 82
FSMoveObjectAsync 83
FSMoveObjectSync 84
FSMoveObjectToTrashAsync 85
FSMoveObjectToTrashSync 86
FSOpenFork 87
FSOpenIterator 88
FSPathCopyObjectAsync 90
FSPathCopyObjectSync 91
FSPathFileOperationCopyStatus 91
FSPathMakeRef 92
FSPathMakeRefWithOptions 93
FSPathMoveObjectAsync 94
FSPathMoveObjectSync 95
FSPathMoveObjectToTrashAsync 96
FSPathMoveObjectToTrashSync 97
FSReadFork 98
FSRefMakePath 99
FSRenameUnicode 100
FSSetCatalogInfo 101
FSSetForkPosition 102
FSSetForkSize 102
FSSetVolumeInfo 103

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSUnlockRange 104
FSUnmountVolumeAsync 105
FSUnmountVolumeSync 105
FSVolumeMount 106
FSWriteFork 106
InvokeFNSubscriptionUPP 108
InvokeFSVolumeEjectUPP 108
InvokeFSVolumeMountUPP 108
InvokeFSVolumeUnmountUPP 109
InvokeIOCompletionUPP 109
NewFNSubscriptionUPP 110
NewFSVolumeEjectUPP 110
NewFSVolumeMountUPP 111
NewFSVolumeUnmountUPP 111
NewIOCompletionUPP 112
PBAllocateForkAsync 112
PBAllocateForkSync 113
PBCatalogSearchAsync 114
PBCatalogSearchSync 116
PBCloseForkAsync 117
PBCloseForkSync 118
PBCloseIteratorAsync 119
PBCloseIteratorSync 119
PBCompareFSRefsAsync 120
PBCompareFSRefsSync 121
PBCreateDirectoryUnicodeAsync 122
PBCreateDirectoryUnicodeSync 123
PBCreateFileUnicodeAsync 124
PBCreateFileUnicodeSync 125
PBCreateForkAsync 126
PBCreateForkSync 127
PBDeleteForkAsync 128
PBDeleteForkSync 129
PBDeleteObjectAsync 130
PBDeleteObjectSync 130
PBExchangeObjectsAsync 131
PBExchangeObjectsSync 132
PBFlushForkAsync 133
PBFlushForkSync 133
PBFlushVolumeAsync 134
PBFlushVolumeSync 134
PBFSCopyFileAsync 135
PBFSCopyFileSync 135
PBGetCatalogInfoAsync 135
PBGetCatalogInfoBulkAsync 136
PBGetCatalogInfoBulkSync 138

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBGetCatalogInfoSync 139
PBGetForkCBInfoAsync 140
PBGetForkCBInfoSync 142
PBGetForkPositionAsync 143
PBGetForkPositionSync 144
PBGetForkSizeAsync 145
PBGetForkSizeSync 145
PBGetVolumeInfoAsync 146
PBGetVolumeInfoSync 148
PBIterateForksAsync 149
PBIterateForksSync 150
PBMakeFSRefUnicodeAsync 151
PBMakeFSRefUnicodeSync 151
PBMoveObjectAsync 152
PBMoveObjectSync 153
PBOpenForkAsync 154
PBOpenForkSync 155
PBOpenIteratorAsync 156
PBOpenIteratorSync 157
PBReadForkAsync 158
PBReadForkSync 159
PBRenameUnicodeAsync 161
PBRenameUnicodeSync 161
PBSetCatalogInfoAsync 162
PBSetCatalogInfoSync 163
PBSetForkPositionAsync 165
PBSetForkPositionSync 165
PBSetForkSizeAsync 166
PBSetForkSizeSync 167
PBSetVolumeInfoAsync 168
PBSetVolumeInfoSync 169
PBWriteForkAsync 170
PBWriteForkSync 171
PBXLockRangeAsync 172
PBXLockRangeSync 173
PBXUnlockRangeAsync 173
PBXUnlockRangeSync 173
Callbacks by Task 174
File Operation Callbacks 174
Miscellaneous Callbacks 174
Callbacks 174
FNSubscriptionProcPtr 174
FSFileOperationStatusProcPtr 175
FSPathFileOperationStatusProcPtr 176
FSVolumeEjectProcPtr 177
FSVolumeMountProcPtr 178

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSVolumeUnmountProcPtr 179
IOCompletionProcPtr 179
Data Types 180
AccessParam 180
AFPAlternateAddress 182
AFPTagData 182
AFPVolMountInfo 183
AFPXVolMountInfo 185
CatPositionRec 187
CInfoPBRec 187
CMovePBRec 188
CntrlParam 189
ConstFSSpecPtr 191
ConstHFSUniStr255Param 191
CopyParam 191
CSParam 193
DirInfo 195
DrvQEl 198
DTPBRec 199
FCBPBRec 202
FIDParam 204
FileParam 205
FNSubscriptionRef 208
FNSubscriptionUPP 208
ForeignPrivParam 208
FSCatalogBulkParam 210
FSCatalogInfo 212
FSCatalogInfoBitmap 214
FSEjectStatus 215
FSFileOperationClientContext 215
FSFileOperationRef 216
FSForkCBInfoParam 216
FSForkInfo 218
FSForkIOParam 219
FSIterator 221
FSMountStatus 221
FSPermissionInfo 222
FSRangeLockParam 222
FSRangeLockParamPtr 222
FSRef 223
FSRefParam 223
FSSearchParams 225
FSSpec 226
FSSpecArrayPtr 227
FSUnmountStatus 228
FSVolumeEjectUPP 228

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSVolumeInfo 228
FSVolumeInfoBitmap 231
FSVolumeInfoParam 231
FSVolumeMountUPP 232
FSVolumeOperation 233
FSVolumeRefNum 233
FSVolumeUnmountUPP 233
GetVolParmsInfoBuffer 233
HFileInfo 235
HFileParam 238
HFSUniStr255 241
HIOParam 241
HParamBlockRec 243
HVolumeParam 245
IOCompletionUPP 247
IOParam 248
MultiDevParam 249
ObjParam 251
ParamBlockRec 252
SlotDevParam 253
VCB 254
VolMountInfoHeader 258
VolumeMountInfoHeader 259
VolumeParam 259
VolumeType 261
WDParam 262
WDPBRec 263
XCInfoPBRec 265
XIOParam 266
XVolumeParam 268
Constants 271
AFP Tag Length Constants 271
AFP Tag Type Constants 272
Allocation Flags 273
AppleShare Volume Signature 274
Authentication Method Constants 274
Cache Constants 275
Catalog Information Bitmap Constants 277
Catalog Information Node Flags 280
Catalog Information Sharing Flags 282
Catalog Search Bits 282
Catalog Search Constants 285
Catalog Search Masks 286
Extended AFP Volume Mounting Information Flag 289
Extended Volume Attributes 289
FCB Flags 293

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

File Access Permission Constants 295
File and Folder Access Privilege Constants 297
File Attribute Constants 300
File Operation Options 303
File Operation Stages 304
File Operation Status Dictionary Keys 305
FNMessage 307
Foreign Privilege Model Constant 307
Group ID Constant 307
Icon Size Constants 307
Icon Type Constants 308
Invalid Volume Reference Constant 310
Iterator Flags 310
kAsyncMountInProgress 311
Notification Subscription Options 311
kHFSCatalogNodeIDsReusedBit 312
Large Volume Constants 312
Mapping Code Constants 312
Path Conversion Options 314
Position Mode Constants 314
Root Directory Constants 315
User ID Constants 315
User Privileges Constants 316
Volume Attribute Constants 317
Volume Control Block Flags 321
Volume Information Attribute Constants 323
Volume Information Bitmap Constants 324
Volume Information Flags 326
Volume Mount Flags 328
Result Codes 329

Appendix A **Deprecated File Manager Functions 341**

Deprecated in Mac OS X v10.4 341
Allocate 341
AllocContig 342
CatMove 344
DirCreate 345
FSClose 345
FSMakeFSSpec 346
FSpCatMove 348
FSpCreate 349
FSpDelete 350
FSpDirCreate 351
FSpExchangeFiles 352
FSpGetFInfo 353

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSpOpenDF 354
FSpOpenRF 355
FSpRename 356
FSpRstFLock 357
FSpSetFInfo 358
FSpSetFLock 358
FSRead 359
FSWrite 360
GetEOF 361
GetFPos 362
GetVRefNum 363
HCreate 363
HDelete 365
HGetFInfo 365
HGetVol 366
HOpen 367
HOpenDF 368
HOpenRF 368
HRename 369
HRstFLock 370
HSetFInfo 371
HSetFLock 372
HSetVol 373
PBAllocateAsync 373
PBAllocateSync 375
PBAllocContigAsync 376
PBAllocContigSync 378
PBCatMoveAsync 379
PBCatMoveSync 380
PBCatSearchAsync 382
PBCatSearchSync 384
PBDirCreateAsync 386
PBDirCreateSync 387
PBDTAddAPPLAsync 388
PBDTAddAPPLSync 389
PBDTAddIconAsync 390
PBDTAddIconSync 391
PBDTCloseDown 392
PBDTDeleteAsync 393
PBDTDeleteSync 394
PBDTFlushAsync 394
PBDTFlushSync 396
PBDTGetAPPLAsync 397
PBDTGetAPPLSync 398
PBDTGetCommentAsync 399
PBDTGetCommentSync 400

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBDTGetIconAsync 401
PBDTGetIconInfoAsync 402
PBDTGetIconInfoSync 403
PBDTGetIconSync 405
PBDTGetInfoAsync 406
PBDTGetInfoSync 407
PBDTGetPath 408
PBDTOpenInform 409
PBDTRemoveAPPLAsync 410
PBDTRemoveAPPLSync 411
PBDTRemoveCommentAsync 411
PBDTRemoveCommentSync 412
PBDTResetAsync 413
PBDTResetSync 414
PBDTSetCommentAsync 415
PBDTSetCommentSync 416
PBExchangeFilesAsync 417
PBExchangeFilesSync 419
PBFlushFileAsync 421
PBFlushFileSync 422
PBGetCatInfoAsync 423
PBGetCatInfoSync 426
PBGetEOFAsync 429
PBGetEOFSync 430
PBGetFCBInfoAsync 431
PBGetFCBInfoSync 432
PBGetForeignPrivsAsync 434
PBGetForeignPrivsSync 435
PBGetFPosAsync 435
PBGetFPosSync 436
PBGetUGEntryAsync 437
PBGetUGEntrySync 437
PBGetXCatInfoAsync 437
PBGetXCatInfoSync 438
PBHCreateAsync 438
PBHCreateSync 439
PBHDeleteAsync 440
PBHDeleteSync 441
PBHGetFInfoAsync 442
PBHGetFInfoSync 444
PBHGetLogInInfoAsync 445
PBHGetLogInInfoSync 446
PBHGetVInfoAsync 447
PBHGetVInfoSync 450
PBHGetVolAsync 453
PBHGetVolSync 454

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHMoveRenameAsync 455
PBHMoveRenameSync 456
PBHOpenAsync 457
PBHOpenDFAsync 458
PBHOpenDFSync 460
PBHOpenRFAsync 461
PBHOpenRFSync 462
PBHOpenSync 463
PBHRenameAsync 465
PBHRenameSync 466
PBHRstFLockAsync 467
PBHRstFLockSync 468
PBHSetFInfoAsync 469
PBHSetFInfoSync 470
PBHSetFLockAsync 470
PBHSetFLockSync 471
PBHSetVolAsync 472
PBHSetVolSync 473
PBLockRangeAsync 474
PBLockRangeSync 476
PBMakeFSSpecAsync 477
PBMakeFSSpecSync 478
PBSetCatInfoAsync 480
PBSetCatInfoSync 481
PBSetEOFAsync 483
PBSetEOFSync 484
PBSetForeignPrivsAsync 485
PBSetForeignPrivsSync 485
PBSetFPosAsync 485
PBSetFPosSync 486
PBSetVInfoAsync 487
PBSetVInfoSync 488
PBShareAsync 489
PBShareSync 490
PBUnlockRangeAsync 490
PBUnlockRangeSync 491
PBUnmountVol 492
PBUnshareAsync 493
PBUnshareSync 494
PBXGetVolInfoAsync 494
PBXGetVolInfoSync 497
SetEOF 499
SetFPos 500
UnmountVol 501
Deprecated in Mac OS X v10.5 502
FlushVol 502

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSpMakeFSRef 503
PBCloseAsync 504
PBCloseSync 505
PBCreateFileIDRefAsync 505
PBCreateFileIDRefSync 506
PBDeleteFileIDRefAsync 507
PBDeleteFileIDRefSync 508
PBFlushVolAsync 509
PBFlushVolSync 510
PBGetVolMountInfo 511
PBGetVolMountInfoSize 512
PBHCopyFileAsync 513
PBHCopyFileSync 514
PBHGetDirAccessAsync 515
PBHGetDirAccessSync 516
PBHGetVolParmsAsync 517
PBHGetVolParmsSync 518
PBHMapIDAsync 519
PBHMapIDSync 520
PBHMapNameAsync 521
PBHMapNameSync 522
PBHOpenDenyAsync 523
PBHOpenDenySync 524
PBHOpenRFDenyAsync 526
PBHOpenRFDenySync 527
PBHSetDirAccessAsync 528
PBHSetDirAccessSync 529
PBMakeFSRefAsync 530
PBMakeFSRefSync 531
PBReadAsync 532
PBReadSync 533
PBResolveFileIDRefAsync 534
PBResolveFileIDRefSync 535
PBVolumeMount 536
PBWaitIOComplete 538
PBWriteAsync 538
PBWriteSync 539

Document Revision History 541

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

File Manager Reference

Framework:	CoreServices/CoreServices.h
Declared in	Files.h HFSVolumes.h

Overview

The File Manager is a core service in Mac OS X that manages the organization, reading, and writing of data located on physical data storage devices such as disk drives. The File Manager provides an abstraction layer that hides lower-level implementation details such as different file systems and volume formats. If you want your application to have the same view of the file system seen in the Mac OS X user interface, the File Manager is an appropriate tool. For example, the File Manager is often used in application frameworks such as Carbon and Cocoa to implement file-related operations.

The File Manager API provides a large number of functions for performing various operations on files, directories, and volumes. The requirements of your application will dictate which of these functions you need to use. Many applications simply need to open files, read and write the data in those files, and then close the files. Other applications might provide more capabilities, such as the ability to copy or move a file to another directory. A few programs, such as the Mac OS X Finder, perform more extensive file operations and hence need to use some of the advanced functions provided by the File Manager.

A number of deprecated functions in the File Manager were inherited from earlier versions of Mac OS and have been carried along to facilitate porting legacy applications to Mac OS X. You should avoid using these deprecated functions. In particular, you should avoid any function or data structure that uses the `FSSpec` data type. This reference document clearly marks every deprecated function and, in most cases, provides a recommended replacement.

Functions by Task

Accessing Information About Files and Directories

`FSGetCatalogInfo` (page 67)
: Returns catalog information about a file or directory. You can use this function to map an `FSRef` to an `FSSpec`.

`PBGetCatalogInfoSync` (page 139)
: Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBGetCatalogInfoAsync` (page 135)
Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

`FSSetCatalogInfo` (page 101)
Sets catalog information about a file or directory.

`PBSetCatalogInfoSync` (page 163)
Sets the catalog information about a file or directory.

`PBSetCatalogInfoAsync` (page 162)
Sets the catalog information about a file or directory.

`FSpGetFInfo` (page 353) Deprecated in Mac OS X v10.4
Obtains the Finder information for a file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`FSpSetFInfo` (page 358) Deprecated in Mac OS X v10.4
Sets the Finder information about a file. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`HGetFInfo` (page 365) Deprecated in Mac OS X v10.4
Obtains the Finder information for a file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`HSetFInfo` (page 371) Deprecated in Mac OS X v10.4
Sets the Finder information for a file. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`PBGetCatInfoAsync` (page 423) Deprecated in Mac OS X v10.4
Returns catalog information about a file or directory. (Deprecated. Use `PBGetCatalogInfoAsync` (page 135) instead.)

`PBGetCatInfoSync` (page 426) Deprecated in Mac OS X v10.4
Returns catalog information about a file or directory. (Deprecated. Use `PBGetCatalogInfoSync` (page 139) instead.)

`PBHGetFInfoAsync` (page 442) Deprecated in Mac OS X v10.4
Obtains information about a file. (Deprecated. Use `PBGetCatalogInfoAsync` (page 135) instead.)

`PBHGetFInfoSync` (page 444) Deprecated in Mac OS X v10.4
Obtains information about a file. (Deprecated. Use `PBGetCatalogInfoSync` (page 139) instead.)

`PBHSetFInfoAsync` (page 469) Deprecated in Mac OS X v10.4
Sets information for a file. (Deprecated. Use `PBSetCatalogInfoAsync` (page 162) instead.)

`PBHSetFInfoSync` (page 470) Deprecated in Mac OS X v10.4
Sets information for a file. (Deprecated. Use `PBSetCatalogInfoSync` (page 163) instead.)

`PBSetCatInfoAsync` (page 480) Deprecated in Mac OS X v10.4
Modifies catalog information for a file or directory. (Deprecated. Use `PBSetCatalogInfoAsync` (page 162) instead.)

`PBSetCatInfoSync` (page 481) Deprecated in Mac OS X v10.4
Modifies catalog information for a file or directory. (Deprecated. Use `PBSetCatalogInfoSync` (page 163) instead.)

Accessing the Desktop Database

`PBDTAddAPPLAsync` (page 388) Deprecated in Mac OS X v10.4
Adds an application to the desktop database. (Deprecated. There is no replacement function.)

`PBDTAddAPPLSync` (page 389) Deprecated in Mac OS X v10.4
Adds an application to the desktop database. (Deprecated. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBDTAddIconAsync (page 390) Deprecated in Mac OS X v10.4

Adds an icon definition to the desktop database. (Deprecated. There is no replacement function.)

PBDTAddIconSync (page 391) Deprecated in Mac OS X v10.4

Adds an icon definition to the desktop database. (Deprecated. There is no replacement function.)

PBDTCloseDown (page 392) Deprecated in Mac OS X v10.4

Closes the desktop database, though your application should never do this itself. (Deprecated. There is no replacement function.)

PBDTDeleteAsync (page 393) Deprecated in Mac OS X v10.4

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTDeleteSync (page 394) Deprecated in Mac OS X v10.4

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

PBDTFlushAsync (page 394) Deprecated in Mac OS X v10.4

Saves your changes to the desktop database. (Deprecated. There is no replacement function.)

PBDTFlushSync (page 396) Deprecated in Mac OS X v10.4

Saves your changes to the desktop database. (Deprecated. There is no replacement function.)

PBDTGetAPPLAsync (page 397) Deprecated in Mac OS X v10.4

Identifies the application that can open a file with a given creator. (Deprecated. There is no replacement function.)

PBDTGetAPPLSync (page 398) Deprecated in Mac OS X v10.4

Identifies the application that can open a file with a given creator. (Deprecated. There is no replacement function.)

PBDTGetCommentAsync (page 399) Deprecated in Mac OS X v10.4

Retrieves the user comments for a file or directory. (Deprecated. There is no replacement function.)

PBDTGetCommentSync (page 400) Deprecated in Mac OS X v10.4

Retrieves the user comments for a file or directory. (Deprecated. There is no replacement function.)

PBDTGetIconAsync (page 401) Deprecated in Mac OS X v10.4

Retrieves an icon definition. (Deprecated. There is no replacement function.)

PBDTGetIconInfoAsync (page 402) Deprecated in Mac OS X v10.4

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated. There is no replacement function.)

PBDTGetIconInfoSync (page 403) Deprecated in Mac OS X v10.4

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated. There is no replacement function.)

PBDTGetIconSync (page 405) Deprecated in Mac OS X v10.4

Retrieves an icon definition. (Deprecated. There is no replacement function.)

PBDTGetInfoAsync (page 406) Deprecated in Mac OS X v10.4

Determines information about the location and size of the desktop database on a particular volume. (Deprecated. There is no replacement function.)

PBDTGetInfoSync (page 407) Deprecated in Mac OS X v10.4

Determines information about the location and size of the desktop database on a particular volume. (Deprecated. There is no replacement function.)

PBDTGetPath (page 408) Deprecated in Mac OS X v10.4

Gets the reference number of the specified desktop database. (Deprecated. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBDTOpenInform` (page 409) Deprecated in Mac OS X v10.4
: Gets the reference number of the specified desktop database, reporting whether the desktop database was empty when it was opened. (Deprecated. There is no replacement function.)

`PBDTRemoveAPPLAsync` (page 410) Deprecated in Mac OS X v10.4
: Removes an application from the desktop database. (Deprecated. There is no replacement function.)

`PBDTRemoveAPPLSync` (page 411) Deprecated in Mac OS X v10.4
: Removes an application from the desktop database. (Deprecated. There is no replacement function.)

`PBDTRemoveCommentAsync` (page 411) Deprecated in Mac OS X v10.4
: Removes a user comment associated with a file or directory from the desktop database. (Deprecated. There is no replacement function.)

`PBDTRemoveCommentSync` (page 412) Deprecated in Mac OS X v10.4
: Removes a user comment associated with a file or directory from the desktop database. (Deprecated. There is no replacement function.)

`PBDTResetAsync` (page 413) Deprecated in Mac OS X v10.4
: Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

`PBDTResetSync` (page 414) Deprecated in Mac OS X v10.4
: Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated. There is no replacement function.)

`PBDTSetCommentAsync` (page 415) Deprecated in Mac OS X v10.4
: Adds a user comment for a file or a directory to the desktop database. (Deprecated. There is no replacement function.)

`PBDTSetCommentSync` (page 416) Deprecated in Mac OS X v10.4
: Adds a user comment for a file or a directory to the desktop database. (Deprecated. There is no replacement function.)

Allocating Storage for Files

`FSAllocateFork` (page 43)
: Allocates space on a volume to an open fork.

`PBAllocateForkSync` (page 113)
: Allocates space on a volume to an open fork.

`PBAllocateForkAsync` (page 112)
: Allocates space on a volume to an open fork.

`Allocate` (page 341) Deprecated in Mac OS X v10.4
: Allocates additional space on a volume to an open file. (Deprecated. Use `FSAllocateFork` (page 43) instead.)

`AllocContig` (page 342) Deprecated in Mac OS X v10.4
: Allocates additional contiguous space on a volume to an open file. (Deprecated. Use `FSAllocateFork` (page 43) instead.)

`PBAllocateAsync` (page 373) Deprecated in Mac OS X v10.4
: Allocates additional space on a volume to an open file. (Deprecated. Use `PBAllocateForkAsync` (page 112) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBAllocateSync` (page 375) Deprecated in Mac OS X v10.4
Allocates additional space on a volume to an open file. (Deprecated. Use `PBAllocateForkSync` (page 113) instead.)

`PBAllocContigAsync` (page 376) Deprecated in Mac OS X v10.4
Allocates additional contiguous space on a volume to an open file. (Deprecated. Use `PBAllocateForkAsync` (page 112) instead.)

`PBAllocContigSync` (page 378) Deprecated in Mac OS X v10.4
Allocates additional contiguous space on a volume to an open file. (Deprecated. Use `PBAllocateForkSync` (page 113) instead.)

Closing Files

`FSCloseFork` (page 47)
Closes an open fork.

`PBCloseForkSync` (page 118)
Closes an open fork.

`PBCloseForkAsync` (page 117)
Closes an open fork.

`PBCloseAsync` (page 504) Deprecated in Mac OS X v10.5
Closes an open file. (Deprecated. Use `PBCloseForkAsync` (page 117) instead.)

`PBCloseSync` (page 505) Deprecated in Mac OS X v10.5
Closes an open file. (Deprecated. Use `PBCloseForkSync` (page 118) instead.)

`FSClose` (page 345) Deprecated in Mac OS X v10.4
Closes an open file. (Deprecated. Use `FSCloseFork` (page 47) instead.)

Comparing File System References

`FSCompareFSRefs` (page 48)
Determines whether two `FSRef` structures refer to the same file or directory.

`PBCompareFSRefsSync` (page 121)
Determines whether two `FSRef` structures refer to the same file or directory.

`PBCompareFSRefsAsync` (page 120)
Determines whether two `FSRef` structures refer to the same file or directory.

Controlling Directory Access

`PBHGetDirAccessAsync` (page 515) Deprecated in Mac OS X v10.5
Returns the access control information for a directory or file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`PBHGetDirAccessSync` (page 516) Deprecated in Mac OS X v10.5
Returns the access control information for a directory or file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHSetDirAccessAsync (page 528) Deprecated in Mac OS X v10.5
Changes the access control information for a directory. (Deprecated. Use FSSetCatalogInfo (page 101) instead.)

PBHSetDirAccessSync (page 529) Deprecated in Mac OS X v10.5
Changes the access control information for a directory. (Deprecated. Use FSSetCatalogInfo (page 101) instead.)

Controlling Login Access

PBHMapIDAsync (page 519) Deprecated in Mac OS X v10.5
Determines the name of a user or group given the user or group ID. (Deprecated. There is no replacement function.)

PBHMapIDSync (page 520) Deprecated in Mac OS X v10.5
Determines the name of a user or group given the user or group ID. (Deprecated. There is no replacement function.)

PBHMapNameAsync (page 521) Deprecated in Mac OS X v10.5
Determines the user ID or group ID from a user or group name. (Deprecated. There is no replacement function.)

PBHMapNameSync (page 522) Deprecated in Mac OS X v10.5
Determines the user ID or group ID from a user or group name. (Deprecated. There is no replacement function.)

PBHGetLogInInfoAsync (page 445) Deprecated in Mac OS X v10.4
Determines the login method used to log on to a particular shared volume. (Deprecated. There is no replacement function.)

Converting Between Paths and FSRef Structures

FSRefMakePath (page 99)
Converts an FSRef structure into a POSIX-style pathname.

FSPathMakeRef (page 92)
Converts a POSIX-style pathname into an FSRef structure.

FSPathMakeRefWithOptions (page 93)
Converts a POSIX-style pathname into an FSRef structure with options.

Copying and Moving Files

PBFSCopyFileSync (page 135)
Duplicates a file and optionally renames it.

PBFSCopyFileAsync (page 135)
Duplicates a file and optionally renames it.

PBHCopyFileAsync (page 513) Deprecated in Mac OS X v10.5
Duplicates a file and optionally renames it. (Deprecated. Use PBFSCopyFileAsync (page 135) instead.)

PBHCopyFileSync (page 514) Deprecated in Mac OS X v10.5
Duplicates a file and optionally renames it. (Deprecated. Use PBFSCopyFileSync (page 135) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHMoveRenameAsync (page 455) Deprecated in Mac OS X v10.4
Moves a file or directory and optionally renames it. (Deprecated. Use FSMoveObjectAsync (page 83) instead.)

PBHMoveRenameSync (page 456) Deprecated in Mac OS X v10.4
Moves a file or directory and optionally renames it. (Deprecated. Use FSMoveObjectSync (page 84) instead.)

Copying and Moving Objects Using Asynchronous High-Level File Operations

FSFileOperationCreate (page 61)
Creates an object that represents an asynchronous file operation.

FSFileOperationCancel (page 60)
Cancels an asynchronous file operation.

FSFileOperationGetTypeID (page 62)
Returns the Core Foundation type identifier for the FSFileOperation opaque type.

FSFileOperationScheduleWithRunLoop (page 62)
Schedules an asynchronous file operation with the specified run loop and mode.

FSFileOperationUnscheduleFromRunLoop (page 63)
Unschedules an asynchronous file operation from the specified run loop and mode.

FSCopyObjectAsync (page 50)
Starts an asynchronous file operation to copy a source object to a destination directory.

FSMoveObjectAsync (page 83)
Starts an asynchronous file operation to move a source object to a destination directory.

FSMoveObjectToTrashAsync (page 85)
Starts an asynchronous file operation to move a source object to the Trash.

FSPathCopyObjectAsync (page 90)
Starts an asynchronous file operation to copy a source object to a destination directory using pathnames.

FSPathMoveObjectAsync (page 94)
Starts an asynchronous file operation to move a source object to a destination directory using pathnames.

FSPathMoveObjectToTrashAsync (page 96)
Starts an asynchronous file operation to move a source object, specified using a pathname, to the Trash.

FSFileOperationCopyStatus (page 61)
Gets a copy of the current status information for an asynchronous file operation.

FSPathFileOperationCopyStatus (page 91)
Gets a copy of the current status information for an asynchronous file operation that uses pathnames.

Copying and Moving Objects Using Synchronous High-Level File Operations

FSCopyObjectSync (page 51)
Copies a source object to a destination directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`FSMoveObjectSync` (page 84)
Moves a source object to a destination directory.

`FSMoveObjectToTrashSync` (page 86)
Moves a source object to the Trash.

`FSPathCopyObjectSync` (page 91)
Copies a source object to a destination directory using pathnames.

`FSPathMoveObjectSync` (page 95)
Moves a source object to a destination directory using pathnames.

`FSPathMoveObjectToTrashSync` (page 97)
Moves a source object, specified using a pathname, to the Trash.

Creating a File System Reference (FSRef)

`FSMakeFSRefUnicode` (page 78)
Constructs an `FSRef` for a file or directory, given a parent directory and a Unicode name.

`PBMakeFSRefUnicodeSync` (page 151)
Constructs an `FSRef` for a file or directory, given a parent directory and a Unicode name.

`PBMakeFSRefUnicodeAsync` (page 151)
Constructs an `FSRef` for a file or directory, given a parent directory and a Unicode name.

`FSpMakeFSRef` (page 503) Deprecated in Mac OS X v10.5
Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated. There is no replacement function.)

`PBMakeFSRefAsync` (page 530) Deprecated in Mac OS X v10.5
Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated. Use `PBMakeFSRefUnicodeAsync` (page 151) instead.)

`PBMakeFSRefSync` (page 531) Deprecated in Mac OS X v10.5
Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated. Use `PBMakeFSRefUnicodeSync` (page 151) instead.)

Creating and Deleting File ID References

`PBCreateFileIDRefAsync` (page 505) Deprecated in Mac OS X v10.5
Establishes a file ID reference for a file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`PBCreateFileIDRefSync` (page 506) Deprecated in Mac OS X v10.5
Establishes a file ID reference for a file. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`PBDeleteFileIDRefAsync` (page 507) Deprecated in Mac OS X v10.5
Deletes a file ID reference. (Deprecated. There is no replacement function.)

`PBDeleteFileIDRefSync` (page 508) Deprecated in Mac OS X v10.5
Deletes a file ID reference. (Deprecated. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Creating and Deleting Named Forks

`FSCreateFork` (page 55)
Creates a named fork for a file or directory.

`PBCreateForkSync` (page 127)
Creates a named fork for a file or directory.

`PBCreateForkAsync` (page 126)
Creates a named fork for a file or directory.

`FSDeleteFork` (page 56)
Deletes a named fork from a file or directory.

`PBDeleteForkSync` (page 129)
Deletes a named fork from a file or directory.

`PBDeleteForkAsync` (page 128)
Deletes a named fork of a file or directory.

Creating Directories

`FSCreateDirectoryUnicode` (page 52)
Creates a new directory (folder) with a Unicode name.

`PBCreateDirectoryUnicodeSync` (page 123)
Creates a new directory (folder) with a Unicode name.

`PBCreateDirectoryUnicodeAsync` (page 122)
Creates a new directory (folder) with a Unicode name.

`DirCreate` (page 345) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `FSCreateDirectoryUnicode` (page 52) instead.)

`FSpDirCreate` (page 351) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `FSCreateDirectoryUnicode` (page 52) instead.)

`PBDirCreateAsync` (page 386) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `PBCreateDirectoryUnicodeAsync` (page 122) instead.)

`PBDirCreateSync` (page 387) Deprecated in Mac OS X v10.4
Creates a new directory. (Deprecated. Use `PBCreateDirectoryUnicodeSync` (page 123) instead.)

Creating File System Specifications

`FSMakeFSSpec` (page 346) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure describing a file or directory. (Deprecated. Use `FSMakeFSRefUnicode` (page 78) instead.)

`PBMakeFSSpecAsync` (page 477) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure for a file or directory. (Deprecated. Use `PBMakeFSRefUnicodeAsync` (page 151) instead.)

`PBMakeFSSpecSync` (page 478) Deprecated in Mac OS X v10.4
Creates an `FSSpec` structure for a file or directory. (Deprecated. Use `PBMakeFSRefUnicodeSync` (page 151) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Creating Files

`FSCreateFileUnicode` (page 54)
Creates a new file with a Unicode name.

`PBCreateFileUnicodeSync` (page 125)
Creates a new file with a Unicode name.

`PBCreateFileUnicodeAsync` (page 124)
Creates a new file with a Unicode name.

`FSpCreate` (page 349) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `FSCreateFileUnicode` (page 54) instead.)

`HCreate` (page 363) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `FSCreateFileUnicode` (page 54) instead.)

`PBHCreateAsync` (page 438) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `PBCreateFileUnicodeAsync` (page 124) instead.)

`PBHCreateSync` (page 439) Deprecated in Mac OS X v10.4
Creates a new file. (Deprecated. Use `PBCreateFileUnicodeSync` (page 125) instead.)

Creating, Calling, and Deleting Universal Procedure Pointers

`NewIOCompletionUPP` (page 112)
Creates a new universal procedure pointer (UPP) to your I/O completion callback function.

`NewFNSubscriptionUPP` (page 110)
Creates a new universal procedure pointer (UPP) to your directory change callback function.

`NewFSVolumeEjectUPP` (page 110)
Creates a new universal procedure pointer (UPP) to your volume ejection callback function.

`NewFSVolumeMountUPP` (page 111)
Creates a new universal procedure pointer (UPP) to your volume mount callback function.

`NewFSVolumeUnmountUPP` (page 111)
Creates a new universal procedure pointer (UPP) to your volume unmount callback function.

`InvokeIOCompletionUPP` (page 109)
Calls your I/O completion callback function.

`InvokeFNSubscriptionUPP` (page 108)
Calls your directory change callback function.

`InvokeFSVolumeEjectUPP` (page 108)
Calls your volume ejection callback function.

`InvokeFSVolumeMountUPP` (page 108)
Calls your volume mount callback function.

`InvokeFSVolumeUnmountUPP` (page 109)
Calls your volume unmount callback function.

`DisposeIOCompletionUPP` (page 39)
Deletes a universal procedure pointer (UPP) to your I/O completion callback function.

`DisposeFNSubscriptionUPP` (page 37)
Deletes a universal procedure pointer (UPP) to your directory change callback function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`DisposeFSVolumeEjectUPP` (page 38)
Deletes a universal procedure pointer (UPP) to your volume ejection callback function.

`DisposeFSVolumeMountUPP` (page 38)
Deletes a universal procedure pointer (UPP) to your volume mount callback function.

`DisposeFSVolumeUnmountUPP` (page 38)
Deletes a universal procedure pointer (UPP) to your volume unmount callback function.

Deleting Files and Directories

`FSDeleteObject` (page 57)
Deletes a file or an empty directory.

`PBDeleteObjectSync` (page 130)
Deletes a file or an empty directory.

`PBDeleteObjectAsync` (page 130)
Deletes a file or an empty directory.

`FSpDelete` (page 350) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use `FSDeleteObject` (page 57) instead.)

`HDelete` (page 365) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use `FSDeleteObject` (page 57) instead.)

`PBHDeleteAsync` (page 440) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use `PBDeleteObjectAsync` (page 130) instead.)

`PBHDeleteSync` (page 441) Deprecated in Mac OS X v10.4
Deletes a file or directory. (Deprecated. Use `PBDeleteObjectSync` (page 130) instead.)

Determining the Unicode Names of the Data and Resource Forks

`FSGetDataForkName` (page 70)
Returns a Unicode string constant for the name of the data fork.

`FSGetResourceForkName` (page 73)
Returns a Unicode string constant for the name of the resource fork.

Exchanging the Contents of Two Files

`FSExchangeObjects` (page 59)
Swaps the contents of two files.

`PBExchangeObjectsSync` (page 132)
Swaps the contents of two files.

`PBExchangeObjectsAsync` (page 131)
Swaps the contents of two files.

`FSpExchangeFiles` (page 352) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use `FSExchangeObjects` (page 59) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBExchangeFilesAsync` (page 417) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use `PBExchangeObjectsAsync` (page 131) instead.)

`PBExchangeFilesSync` (page 419) Deprecated in Mac OS X v10.4
Exchanges the data stored in two files on the same volume. (Deprecated. Use `PBExchangeObjectsSync` (page 132) instead.)

Getting and Setting Volume Information

`FSGetVolumeInfo` (page 74)
Returns information about a volume.

`PBGetVolumeInfoSync` (page 148)
Returns information about a volume.

`PBGetVolumeInfoAsync` (page 146)
Returns information about a volume.

`FSSetVolumeInfo` (page 103)
Sets information about a volume.

`PBSetVolumeInfoSync` (page 169)
Sets information about a volume.

`PBSetVolumeInfoAsync` (page 168)
Sets information about a volume.

`FSCopyDiskIDForVolume` (page 49)
Returns a copy of the disk ID for a volume.

`FSCopyURLForVolume` (page 51)
Returns a copy of the URL for a volume.

`GetVRefNum` (page 363) Deprecated in Mac OS X v10.4
Gets a volume reference number from a file reference number. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`PBHGetVInfoAsync` (page 447) Deprecated in Mac OS X v10.4
Gets detailed information about a volume. (Deprecated. Use `PBGetVolumeInfoAsync` (page 146) instead.)

`PBHGetVInfoSync` (page 450) Deprecated in Mac OS X v10.4
Gets detailed information about a volume. (Deprecated. Use `PBGetVolumeInfoSync` (page 148) instead.)

`PBSetVInfoAsync` (page 487) Deprecated in Mac OS X v10.4
Changes information about a volume. (Deprecated. Use `PBSetVolumeInfoAsync` (page 168) instead.)

`PBSetVInfoSync` (page 488) Deprecated in Mac OS X v10.4
Changes information about a volume. (Deprecated. Use `PBSetVolumeInfoSync` (page 169) instead.)

`PBXGetVolInfoAsync` (page 494) Deprecated in Mac OS X v10.4
Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated. Use `FSGetVolumeInfo` (page 74) instead.)

`PBXGetVolInfoSync` (page 497) Deprecated in Mac OS X v10.4
Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated. Use `FSGetVolumeInfo` (page 74) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Getting Volume Attributes

`FSGetVolumeParms` (page 76)
Retrieves information about the characteristics of a volume.

`PBHGetVolParmsAsync` (page 517) Deprecated in Mac OS X v10.5
Returns information about the characteristics of a volume. (Deprecated. Use `FSGetVolumeParms` (page 76) instead.)

`PBHGetVolParmsSync` (page 518) Deprecated in Mac OS X v10.5
Returns information about the characteristics of a volume. (Deprecated. Use `FSGetVolumeParms` (page 76) instead.)

Iterating Over Named Forks

`FSIterateForks` (page 77)
Determines the name and size of every named fork belonging to a file or directory.

`PBIterateForksSync` (page 150)
Determines the name and size of every named fork belonging to a file or directory.

`PBIterateForksAsync` (page 149)
Determines the name and size of every named fork belonging to a file or directory.

Locking and Unlocking File Ranges

`FSLockRange` (page 78)
Locks a range of bytes of the specified fork.

`PBXLockRangeSync` (page 173)
Locks a range of bytes of the specified fork.

`PBXLockRangeAsync` (page 172)
Locks a range of bytes of the specified fork.

`FSUnlockRange` (page 104)
Unlocks a range of bytes of the specified fork.

`PBXUnlockRangeSync` (page 173)
Unlocks a range of bytes of the specified fork.

`PBXUnlockRangeAsync` (page 173)
Unlocks a range of bytes of the specified fork.

`PBLockRangeAsync` (page 474) Deprecated in Mac OS X v10.4
Locks a portion of a file. (Deprecated. Use `PBXLockRangeAsync` (page 172) instead.)

`PBLockRangeSync` (page 476) Deprecated in Mac OS X v10.4
Locks a portion of a file. (Deprecated. Use `PBXLockRangeSync` (page 173) or `FSLockRange` (page 78) instead.)

`PBUnlockRangeAsync` (page 490) Deprecated in Mac OS X v10.4
Unlocks a portion of a file. (Deprecated. Use `PBXUnlockRangeAsync` (page 173) instead.)

`PBUnlockRangeSync` (page 491) Deprecated in Mac OS X v10.4
Unlocks a portion of a file. (Deprecated. Use `PBXUnlockRangeSync` (page 173) or `FSUnlockRange` (page 104) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Locking and Unlocking Files and Directories

`FSpRstFLock` (page 357) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`FSpSetFLock` (page 358) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`HRstFLock` (page 370) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`HSetFLock` (page 372) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `FSSetCatalogInfo` (page 101) instead.)

`PBHRstFLockAsync` (page 467) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `PBSetCatalogInfoAsync` (page 162) instead.)

`PBHRstFLockSync` (page 468) Deprecated in Mac OS X v10.4
Unlocks a file or directory. (Deprecated. Use `PBSetCatalogInfoSync` (page 163) instead.)

`PBHSetFLockAsync` (page 470) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `PBSetCatalogInfoAsync` (page 162) instead.)

`PBHSetFLockSync` (page 471) Deprecated in Mac OS X v10.4
Locks a file or directory. (Deprecated. Use `PBSetCatalogInfoSync` (page 163) instead.)

Manipulating File and Fork Size

`FSGetForkSize` (page 73)
Returns the size of an open fork.

`PBGetForkSizeSync` (page 145)
Returns the size of an open fork.

`PBGetForkSizeAsync` (page 145)
Returns the size of an open fork.

`FSSetForkSize` (page 102)
Changes the size of an open fork.

`PBSetForkSizeSync` (page 167)
Changes the size of an open fork.

`PBSetForkSizeAsync` (page 166)
Changes the size of an open fork.

`GetEOF` (page 361) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `FSGetForkSize` (page 73) instead.)

`PBGetEOFAsync` (page 429) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `PBGetForkSizeAsync` (page 145) instead.)

`PBGetEOFSync` (page 430) Deprecated in Mac OS X v10.4
Determines the current logical size of an open file. (Deprecated. Use `PBGetForkSizeSync` (page 145) instead.)

`PBSetEOFAsync` (page 483) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use `PBSetForkSizeAsync` (page 166) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBSetEOFSync (page 484) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use PBSetForkSizeSync (page 167) instead.)

SetEOF (page 499) Deprecated in Mac OS X v10.4
Sets the logical size of an open file. (Deprecated. Use FSSetForkSize (page 102) instead.)

Manipulating File Position

FSGetForkPosition (page 72)
Returns the current position of an open fork.

PBGetForkPositionSync (page 144)
Returns the current position of an open fork.

PBGetForkPositionAsync (page 143)
Returns the current position of an open fork.

FSSetForkPosition (page 102)
Sets the current position of an open fork.

PBSetForkPositionSync (page 165)
Sets the current position of an open fork.

PBSetForkPositionAsync (page 165)
Sets the current position of an open fork.

GetFPos (page 362) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use FSGetForkPosition (page 72) instead.)

PBGetFPosAsync (page 435) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use PBGetForkPositionAsync (page 143) instead.)

PBGetFPosSync (page 436) Deprecated in Mac OS X v10.4
Returns the current position of the file mark. (Deprecated. Use PBGetForkPositionSync (page 144) instead.)

PBSetFPosAsync (page 485) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use PBSetForkPositionAsync (page 165) instead.)

PBSetFPosSync (page 486) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use PBSetForkPositionSync (page 165) instead.)

SetFPos (page 500) Deprecated in Mac OS X v10.4
Sets the position of the file mark. (Deprecated. Use FSSetForkPosition (page 102) instead.)

Manipulating the Default Volume

HGetVol (page 366) Deprecated in Mac OS X v10.4
Determines the current default volume and default directory. (Deprecated. There is no replacement function.)

HSetVol (page 373) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

PBHGetVolAsync (page 453) Deprecated in Mac OS X v10.4
Determines the default volume and default directory. (Deprecated. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBHGetVolSync` (page 454) Deprecated in Mac OS X v10.4
Determines the default volume and default directory. (Deprecated. There is no replacement function.)

`PBHSetVolAsync` (page 472) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

`PBHSetVolSync` (page 473) Deprecated in Mac OS X v10.4
Sets the default volume and the default directory. (Deprecated. There is no replacement function.)

Mounting and Unmounting Volumes

`FSMountLocalVolumeSync` (page 80)
Mounts a volume.

`FSMountServerVolumeSync` (page 82)
Mounts a server volume.

`FSUnmountVolumeSync` (page 105)
Unmounts a volume.

`FSEjectVolumeSync` (page 59)
Ejects a volume.

`FSCreateVolumeOperation` (page 56)
Returns an `FSVolumeOperation` which can be used for an asynchronous volume operation.

`FSCancelVolumeOperation` (page 44)
Cancels an outstanding asynchronous volume mounting operation.

`FSDisposeVolumeOperation` (page 57)
Releases the memory associated with a volume operation.

`FSMountLocalVolumeAsync` (page 79)
Mounts a volume asynchronously.

`FSMountServerVolumeAsync` (page 80)
Mounts a server volume asynchronously.

`FSUnmountVolumeAsync` (page 105)
Unmounts a volume asynchronously.

`FSEjectVolumeAsync` (page 58)
Asynchronously ejects a volume.

`FSGetAsyncMountStatus` (page 65)
Returns the current status of an asynchronous mount operation.

`FSGetAsyncUnmountStatus` (page 66)
Returns the current status of an asynchronous unmount operation.

`FSGetAsyncEjectStatus` (page 65)
Returns the current status of an asynchronous eject operation.

`PBUnmountVol` (page 492) Deprecated in Mac OS X v10.4
Unmounts a volume. (Deprecated. Use `FSEjectVolumeSync` (page 59) or `FSUnmountVolumeSync` (page 105) instead.)

`UnmountVol` (page 501) Deprecated in Mac OS X v10.4
Unmounts a volume that isn't currently being used. (Deprecated. Use `FSUnmountVolumeSync` (page 105) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Mounting Remote Volumes

`FSGetVolumeMountInfoSize` (page 76)
Determines the size of the mounting information associated with the specified volume.

`FSGetVolumeMountInfo` (page 76)
Retrieves the mounting information associated with the specified volume.

`FSVolumeMount` (page 106)
Mounts a volume using the specified mounting information.

`PBGetVolMountInfo` (page 511) Deprecated in Mac OS X v10.5
Retrieves a record containing all the information needed to mount a volume, except for passwords. (Deprecated. Use `FSVolumeMount` (page 106) instead.)

`PBGetVolMountInfoSize` (page 512) Deprecated in Mac OS X v10.5
Determines how much space to allocate for a volume mounting information structure. (Deprecated. Use `FSVolumeMount` (page 106) instead.)

`PBVolumeMount` (page 536) Deprecated in Mac OS X v10.5
Mounts a volume. (Deprecated. Use `FSVolumeMount` (page 106) instead.)

Moving and Renaming Files or Directories

`FSMoveObject` (page 82)
Moves a file or directory into a different directory.

`PBMoveObjectSync` (page 153)
Moves a file or directory into a different directory.

`PBMoveObjectAsync` (page 152)
Moves a file or directory into a different directory.

`FSRenameUnicode` (page 100)
Renames a file or folder.

`PBRenameUnicodeSync` (page 161)
Renames a file or folder.

`PBRenameUnicodeAsync` (page 161)
Renames a file or folder.

`CatMove` (page 344) Deprecated in Mac OS X v10.4
Moves files or directories from one directory to another on the same volume. (Deprecated. Use `FSMoveObject` (page 82) instead.)

`FSpCatMove` (page 348) Deprecated in Mac OS X v10.4
Moves a file or directory from one location to another on the same volume. (Deprecated. Use `FSMoveObject` (page 82) instead.)

`FSpRename` (page 356) Deprecated in Mac OS X v10.4
Renames a file or directory. (Deprecated. Use `FSRenameUnicode` (page 100) instead.)

`HRename` (page 369) Deprecated in Mac OS X v10.4
Renames a file, directory, or volume. (Deprecated. Use `FSRenameUnicode` (page 100) instead.)

`PBCatMoveAsync` (page 379) Deprecated in Mac OS X v10.4
Moves files or directories from one directory to another on the same volume. (Deprecated. Use `PBMoveObjectAsync` (page 152) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBCatMoveSync (page 380) Deprecated in Mac OS X v10.4
: Moves files or directories from one directory to another on the same volume. (Deprecated. Use PBMoveObjectSync (page 153) instead.)

PBHRenameAsync (page 465) Deprecated in Mac OS X v10.4
: Renames a file, directory, or volume. (Deprecated. Use PBRenameUnicodeAsync (page 161) instead.)

PBHRenameSync (page 466) Deprecated in Mac OS X v10.4
: Renames a file, directory, or volume. (Deprecated. Use PBRenameUnicodeSync (page 161) instead.)

Obtaining File and Directory Information Using a Catalog Iterator on HFS Plus Volumes

FSGetCatalogInfoBulk (page 68)
: Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

PBGetCatalogInfoBulkSync (page 138)
: Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

PBGetCatalogInfoBulkAsync (page 136)
: Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

Obtaining File Control Block Information

PBGetFCBInfoAsync (page 431) Deprecated in Mac OS X v10.4
: Gets information about an open file from the file control block. (Deprecated. Use PBGetForkCBInfoAsync (page 140) instead.)

PBGetFCBInfoSync (page 432) Deprecated in Mac OS X v10.4
: Gets information about an open file from the file control block. (Deprecated. Use PBGetForkCBInfoSync (page 142) instead.)

Obtaining Fork Control Block Information

FSGetForkCBInfo (page 71)
: Returns information about a specified open fork, or about all open forks.

PBGetForkCBInfoSync (page 142)
: Returns information about a specified open fork, or about all open forks.

PBGetForkCBInfoAsync (page 140)
: Returns information about a specified open fork, or about all open forks.

Opening Files

FSOpenFork (page 87)
: Opens any fork of a file or directory for streaming access.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBOpenForkSync (page 155)
Opens any fork of a file or directory for streaming access.

PBOpenForkAsync (page 154)
Opens any fork of a file or directory for streaming access.

FSpOpenDF (page 354) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 87) instead.)

FSpOpenRF (page 355) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use FSOpenFork (page 87) instead.)

HOpen (page 367) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 87) instead.)

HOpenDF (page 368) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use FSOpenFork (page 87) instead.)

HOpenRF (page 368) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use FSOpenFork (page 87) instead.)

PBHOpenAsync (page 457) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkAsync (page 154) instead.)

PBHOpenDFAsync (page 458) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkAsync (page 154) instead.)

PBHOpenDFSync (page 460) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkSync (page 155) instead.)

PBHOpenRFAsync (page 461) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use PBOpenForkAsync (page 154) instead.)

PBHOpenRFSync (page 462) Deprecated in Mac OS X v10.4
Opens the resource fork of a file. (Deprecated. Use PBOpenForkSync (page 155) instead.)

PBHOpenSync (page 463) Deprecated in Mac OS X v10.4
Opens the data fork of a file. (Deprecated. Use PBOpenForkSync (page 155) instead.)

Opening Files While Denying Access

PBHOpenDenyAsync (page 523) Deprecated in Mac OS X v10.5
Opens a file's data fork using the access deny modes. (Deprecated. Use PBOpenForkAsync (page 154) with deny modes in the permissions field.)

PBHOpenDenySync (page 524) Deprecated in Mac OS X v10.5
Opens a file's data fork using the access deny modes. (Deprecated. Use PBOpenForkSync (page 155) with deny modes in the permissions field.)

PBHOpenRFDenyAsync (page 526) Deprecated in Mac OS X v10.5
Opens a file's resource fork using the access deny modes. (Deprecated. Use PBOpenForkAsync (page 154) with deny modes in the permissions field.)

PBHOpenRFDenySync (page 527) Deprecated in Mac OS X v10.5
Opens a file's resource fork using the access deny modes. (Deprecated. Use PBOpenForkSync (page 155) with deny modes in the permissions field.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Reading and Writing Files

`FSReadFork` (page 98)
Reads data from an open fork.

`PBReadForkSync` (page 159)
Reads data from an open fork.

`PBReadForkAsync` (page 158)
Reads data from an open fork.

`FSWriteFork` (page 106)
Writes data to an open fork.

`PBWriteForkSync` (page 171)
Writes data to an open fork.

`PBWriteForkAsync` (page 170)
Writes data to an open fork.

`PBReadAsync` (page 532) Deprecated in Mac OS X v10.5
Reads any number of bytes from an open file. (Deprecated. Use `PBReadForkAsync` (page 158) instead.)

`PBReadSync` (page 533) Deprecated in Mac OS X v10.5
Reads any number of bytes from an open file. (Deprecated. Use `PBReadForkSync` (page 159) instead.)

`PBWriteAsync` (page 538) Deprecated in Mac OS X v10.5
Writes any number of bytes to an open file. (Deprecated. Use `PBWriteForkAsync` (page 170) instead.)

`PBWriteSync` (page 539) Deprecated in Mac OS X v10.5
Writes any number of bytes to an open file. (Deprecated. Use `PBWriteForkSync` (page 171) instead.)

`FSRead` (page 359) Deprecated in Mac OS X v10.4
Reads any number of bytes from an open file. (Deprecated. Use `FSReadFork` (page 98) instead.)

`FSWrite` (page 360) Deprecated in Mac OS X v10.4
Writes any number of bytes to an open file. (Deprecated. Use `FSWriteFork` (page 106) instead.)

Resolving File ID References

`PBResolveFileIDRefAsync` (page 534) Deprecated in Mac OS X v10.5
Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

`PBResolveFileIDRefSync` (page 535) Deprecated in Mac OS X v10.5
Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated. Use `FSGetCatalogInfo` (page 67) instead.)

Searching a Volume

`PBCatSearchAsync` (page 382) Deprecated in Mac OS X v10.4
Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated. Use `PBCatalogSearchAsync` (page 114) instead.)

`PBCatSearchSync` (page 384) Deprecated in Mac OS X v10.4
Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated. Use `PBCatalogSearchSync` (page 116) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Searching a Volume Using a Catalog Iterator

`FSOpenIterator` (page 88)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`PBOpenIteratorSync` (page 157)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`PBOpenIteratorAsync` (page 156)
Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

`FSCatalogSearch` (page 45)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`PBCatalogSearchSync` (page 116)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`PBCatalogSearchAsync` (page 114)
Searches for objects traversed by a catalog iterator that match a given set of criteria.

`FSCloseIterator` (page 48)
Closes a catalog iterator.

`PBCloseIteratorSync` (page 119)
Closes a catalog iterator.

`PBCloseIteratorAsync` (page 119)
Closes a catalog iterator.

Updating Files

`FSFlushFork` (page 64)
Causes all data written to an open fork to be written to disk.

`PBFlushForkSync` (page 133)
Causes all data written to an open fork to be written to disk.

`PBFlushForkAsync` (page 133)
Causes all data written to an open fork to be written to disk.

`PBFlushFileAsync` (page 421) Deprecated in Mac OS X v10.4
Writes the contents of a file's access path buffer to the disk. (Deprecated. Use `PBFlushForkAsync` (page 133) instead.)

`PBFlushFileSync` (page 422) Deprecated in Mac OS X v10.4
Writes the contents of a file's access path buffer to the disk. (Deprecated. Use `PBFlushForkSync` (page 133) instead.)

Updating Volumes

`FSFlushVolume` (page 64)
For the specified volume, writes all open and modified files in the current process to permanent storage.

`PBFlushVolumeSync` (page 134)
For the specified volume, writes all open and modified files in the current process to permanent storage.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBFlushVolumeAsync` (page 134)
For the specified volume, writes all open and modified files in the current process to permanent storage.

`FlushVol` (page 502) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `FSFlushVolume` (page 64) instead.)

`PBFlushVolAsync` (page 509) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `PBFlushVolumeAsync` (page 134) instead.)

`PBFlushVolSync` (page 510) Deprecated in Mac OS X v10.5
Writes the contents of the volume buffer and updates information about the volume. (Deprecated. Use `PBFlushVolumeSync` (page 134) instead.)

Using Change Notifications

`FNNotify` (page 40)
Broadcasts notification of changes to the specified directory.

`FNNotifyAll` (page 40)
Broadcasts notification of changes to the filesystem.

`FNNotifyByPath` (page 41)
Broadcasts notification of changes to the specified directory.

`FNSubscribe` (page 41)
Subscribes to change notifications for the specified directory.

`FNSubscribeByPath` (page 42)
Subscribes to change notifications for the specified directory.

`FNUnsubscribe` (page 43)
Releases a subscription which is no longer needed.

`FNGetDirectoryForSubscription` (page 39)
Fetches the directory for which this subscription was originally entered.

Not Recommended

This section lists functions that are not recommended and you should no longer use.

`PBWaitIOComplete` (page 538) Deprecated in Mac OS X v10.5
Keeps the system idle until either an interrupt occurs or the specified timeout value is reached. (Deprecated. There is no replacement function.)

`PBGetForeignPrivsAsync` (page 434) Deprecated in Mac OS X v10.4
Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBGetForeignPrivsSync` (page 435) Deprecated in Mac OS X v10.4
Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBGetUGEntryAsync` (page 437) Deprecated in Mac OS X v10.4
Gets a user or group entry from the list of User and Group names and IDs on the local file server. (Deprecated. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`PBGetUGEntrySync` (page 437) Deprecated in Mac OS X v10.4
: Gets a user or group entry from the list of User and Group names and IDs on a local file server. (Deprecated. There is no replacement function.)

`PBGetXCatInfoAsync` (page 437) Deprecated in Mac OS X v10.4
: Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated. There is no replacement function.)

`PBGetXCatInfoSync` (page 438) Deprecated in Mac OS X v10.4
: Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated. There is no replacement function.)

`PBHGetLogInInfoSync` (page 446) Deprecated in Mac OS X v10.4
: Determines the login method used to log on to a particular shared volume. (Deprecated. There is no replacement function.)

`PBSetForeignPrivsAsync` (page 485) Deprecated in Mac OS X v10.4
: Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBSetForeignPrivsSync` (page 485) Deprecated in Mac OS X v10.4
: Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated. There is no replacement function.)

`PBShareAsync` (page 489) Deprecated in Mac OS X v10.4
: Establishes a local volume or directory as a share point. (Deprecated. There is no replacement function.)

`PBShareSync` (page 490) Deprecated in Mac OS X v10.4
: Establishes a local volume or directory as a share point. (Deprecated. There is no replacement function.)

`PBUnshareAsync` (page 493) Deprecated in Mac OS X v10.4
: Makes a share point unavailable on the network. (Deprecated. There is no replacement function.)

`PBUnshareSync` (page 494) Deprecated in Mac OS X v10.4
: Makes a share point unavailable on the network. (Deprecated. There is no replacement function.)

Functions

DisposeFNSubscriptionUPP

Deletes a universal procedure pointer (UPP) to your directory change callback function.

```
void DisposeFNSubscriptionUPP (
   FNSubscriptionUPP userUPP
);
```

Parameters

userUPP
: The UPP to delete.

Discussion

You should use this function to delete the UPP after the File Manager is finished calling your directory change callback function.

Availability

Available in Mac OS X v10.1 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

DisposeFSVolumeEjectUPP

Deletes a universal procedure pointer (UPP) to your volume ejection callback function.

```
void DisposeFSVolumeEjectUPP (
   FSVolumeEjectUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your volume ejection callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

DisposeFSVolumeMountUPP

Deletes a universal procedure pointer (UPP) to your volume mount callback function.

```
void DisposeFSVolumeMountUPP (
   FSVolumeMountUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion
You should use this function to delete the UPP after the File Manager is finished calling your volume mount callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

DisposeFSVolumeUnmountUPP

Deletes a universal procedure pointer (UPP) to your volume unmount callback function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void DisposeFSVolumeUnmountUPP (
   FSVolumeUnmountUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion

You should use this function to delete the UPP after the File Manager is finished calling your volume unmount callback function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

DisposeIOCompletionUPP

Deletes a universal procedure pointer (UPP) to your I/O completion callback function.

```
void DisposeIOCompletionUPP (
   IOCompletionUPP userUPP
);
```

Parameters

userUPP

The UPP to delete.

Discussion

You should use this function to delete the UPP after the File Manager is finished calling your I/O completion callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FNGetDirectoryForSubscription

Fetches the directory for which this subscription was originally entered.

```
OSStatus FNGetDirectoryForSubscription (
   FNSubscriptionRef subscription,
   FSRef *ref
);
```

Parameters

subscription

The subscription previously returned from the functions `FNSubscribe` or `FNSubscribeByPath`.

ref

On return, a file system reference to the directory for which this subscription was created.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
There is no path variant because paths are fragile, and the path may have changed. If the caller does not care about this subtlety, she can call `FSRefMakePath` to get a path from the returned reference.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

FNNotify

Broadcasts notification of changes to the specified directory.

```
OSStatus FNNotify (
   const FSRef *ref,
   FNMessage message,
   OptionBits flags
);
```

Parameters

ref
A file system reference describing the directory for which to broadcast the notification.

message
An indication of what happened to the target directory.

flags
Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FNNotifyAll

Broadcasts notification of changes to the filesystem.

```
OSStatus FNNotifyAll (
   FNMessage message,
   OptionBits flags
);
```

Parameters

message
An indication of what happened.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

flags

Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function should only be used by installers or programs which make lots of changes and only send one broadcast.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FNNotifyByPath

Broadcasts notification of changes to the specified directory.

```
OSStatus FNNotifyByPath (
   const UInt8 *path,
   FNMessage message,
   OptionBits flags
);
```

Parameters

path

The path to the directory for which to broadcast the notification.

message

An indication of what happened to the target directory.

flags

Options regarding the delivery of the notification. Specify `kNilOptions` for the default behavior.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FNSubscribe

Subscribes to change notifications for the specified directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FNSubscribe (
   const FSRef *directoryRef,
   FNSubscriptionUPP callback,
   void *refcon,
   OptionBits flags,
   FNSubscriptionRef *subscription
);
```

Parameters

directoryRef

A file system reference describing the directory for which the caller wants notifications.

callback

A pointer to the function to call when a notification arrives.

refcon

A pointer to user state carried with the subscription.

flags

Specify `kNilOptions`, or one of the options described in "Notification Subscription Options" (page 311).

subscription

A subscription token for subsequent query or unsubscription.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Files.h`

FNSubscribeByPath

Subscribes to change notifications for the specified directory.

```
OSStatus FNSubscribeByPath (
   const UInt8 *directoryPath,
   FNSubscriptionUPP callback,
   void *refcon,
   OptionBits flags,
   FNSubscriptionRef *subscription
);
```

Parameters

directoryPath

A path to the directory for which the caller wants notifications.

callback

The function to call when a notification arrives.

refcon

A pointer to the user state carried with the subscription.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

flags

Specify `kNilOptions`, or one of the options described in "Notification Subscription Options" (page 311).

subscription

A subscription token for subsequent query or unsubscription.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

FNUnsubscribe

Releases a subscription which is no longer needed.

```
OSStatus FNUnsubscribe (
   FNSubscriptionRef subscription
);
```

Parameters

subscription

A subscription previously returned from the `FNSubscribe` or`FNSubscribeByPath` functions.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

FSAllocateFork

Allocates space on a volume to an open fork.

```
OSErr FSAllocateFork (
   FSIORefNum forkRefNum,
   FSAllocationFlags flags,
   UInt16 positionMode,
   SInt64 positionOffset,
   UInt64 requestCount,
   UInt64 *actualCount
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 87) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

flags

A constant indicating how the new space should be allocated. See "Allocation Flags" (page 273) for a description of the constants which you can use in this parameter.

positionMode

A constant specifying the base location for the start of the allocation. See "Position Mode Constants" (page 314) for more information on the constants which you can use to specify the base location.

positionOffset

The offset from the base location of the start of the allocation.

requestCount

The number of bytes to allocate.

actualCount

On return, a pointer to the number of bytes actually allocated to the file. The value returned in here may be smaller than the number specified in the `requestCount` parameter if some of the space was already allocated. The value pointed to by the `actualCount` parameter does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The `actualCount` output is optional if you don't want the number of allocated bytes returned, set `actualCount` to `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSAllocateFork` function attempts to allocate `requestCount` bytes of physical storage starting at the offset specified by the `positionMode` and `positionOffset` parameters. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using `FSCloseFork` (page 47) , `PBCloseForkSync` (page 118) , or `PBCloseForkAsync` (page 117) ; or when the fork is flushed, using `FSFlushFork` (page 64) , `PBFlushForkSync` (page 133) , or `PBFlushForkAsync` (page 133).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

FSCancelVolumeOperation

Cancels an outstanding asynchronous volume mounting operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSCancelVolumeOperation (
   FSVolumeOperation volumeOp
);
```

Parameters

volumeOp

The asynchronous volume operation to cancel.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Special Considerations

This function currently is only supported for server mounts.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSCatalogSearch

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
OSErr FSCatalogSearch (
   FSIterator iterator,
   const FSSearchParams *searchCriteria,
   ItemCount maximumObjects,
   ItemCount *actualObjects,
   Boolean *containerChanged,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfos,
   FSRef *refs,
   FSSpecPtr specs,
   HFSUniStr255 *names
);
```

Parameters

iterator

The iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchCriteria` parameter. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88), or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 221) for more information on the `FSIterator` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

searchCriteria

A pointer to a structure containing the search criteria.

You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 212) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 384) and `PBCatSearchAsync` (page 382) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 286) for a description of the search bits.

There are a few new search criteria supported by `FSCatalogSearch` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants described in "Catalog Search Constants" (page 285).

If the `searchTime` field of this structure is non-zero, it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumObjects` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search.

If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this parameter to point to `FSCatalogInfo` structures containing the values to match against.

See `FSSearchParams` (page 225) for a description of the `FSSearchParams` data type.

maximumObjects

The maximum number of items to return for this call.

actualObjects

On return, a pointer to the actual number of items found for this call.

containerChanged

On return, a pointer to a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `FSCatalogSearch` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

This parameter is optional if you don't want this information, pass a `NULL` pointer.

whichInfo

A bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this parameter. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this parameter.

catalogInfos

A pointer to an array of catalog information structures; one for each found item. On input, the `catalogInfos` parameter should point to an array of `maximumObjects` catalog information structures.

This parameter is optional; if you do not wish any catalog information returned, pass `NULL` here.

See `FSCatalogInfo` (page 212) for a description of the `FSCatalogInfo` data type.

refs

A pointer to an array of `FSRef` structures; one for each returned item. If you want an `FSRef` for each item found, set this parameter to point to an array of `maximumObjectsFSRef` structures. Otherwise, set it to `NULL`. See `FSRef` (page 223) for a description of the `FSRef` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

specs

names

A pointer to an array of filenames; one for each returned item. If you want the Unicode filename for each item found, set this parameter to point to an array of `maximumObjectsHFSUniStr255` structures. Otherwise, set it to `NULL`. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). When the entire volume has been searched, `errFSNoMoreItems` is returned.

Discussion
A single search may span more than one call to `FSCatalogSearch`. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (`searchTime`) has been reached or `maximumObjects` items have been returned. If the search is not completed, you can continue the search by making another call to `FSCatalogSearch` and passing the updated iterator returned by the previous call in the `iterator` parameter.

Before calling this function, you should determine that it is present, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSCloseFork

Closes an open fork.

```
OSErr FSCloseFork (
    FSIORefNum forkRefNum
);
```

Parameters

forkRefNum

The reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSCloseFork` function causes all data written to the fork to be written to disk, in the same manner as the `FSFlushFork` (page 64) function, before it closes the fork.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioCDSample
Cocoa PDE with Carbon Printing
CoreTextTest
FSCopyObject

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSCreateFileAndOpenForkUnicode

Declared In
`Files.h`

FSCloseIterator

Closes a catalog iterator.

```
OSErr FSCloseIterator (
   FSIterator iterator
);
```

Parameters

iterator

The catalog iterator to be closed. `FSCloseIterator` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function releases memory and other system resources used by the iterator. The iterator becomes invalid.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject
FSMegaInfo
MFSLives
MoreSCF

Declared In
`Files.h`

FSCompareFSRefs

Determines whether two `FSRef` structures refer to the same file or directory.

```
OSErr FSCompareFSRefs (
   const FSRef *ref1,
   const FSRef *ref2
);
```

Parameters

ref1

A pointer to the first `FSRef` to compare. For a description of the `FSRef` data type, see `FSRef` (page 223).

ref2

A pointer to the second `FSRef` to compare.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the two `FSRef` structures refer to the same file or directory, then `noErr` is returned. If they refer to objects on different volumes, then `diffVolErr` is returned. If they refer to different files or directories on the same volume, then `errFSRefsDifferent` is returned. This function may return other errors, including `nsvErr`, `fnfErr`, `dirNFErr`, and `volOffLinErr`.

Discussion
You must use `FSCompareFSRefs`, or one of the corresponding parameter block functions, `PBCompareFSRefsSync` (page 121) and `PBCompareFSRefsAsync` (page 120) , to compare `FSRef` structures. It is not possible to compare the `FSRef` structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

Some volume formats may be able to tell that two `FSRef` structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return `errFSRefsDifferent` even if one or both objects no longer exist. Similarly, if the `FSRef` structures are for objects on different volumes, the File Manager will return `diffVolErr` even if one or both volumes are no longer mounted.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

FSCopyDiskIDForVolume

Returns a copy of the disk ID for a volume.

```
OSStatus FSCopyDiskIDForVolume (
    FSVolumeRefNum vRefNum,
    CFStringRef *diskID
);
```

Parameters

vRefNum
The volume reference number of the target volume.

diskID
A pointer to a Core Foundation string. On return, the string contains the disk ID associated with the target volume. The caller is responsible for releasing the string.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSCopyObjectAsync

Starts an asynchronous file operation to copy a source object to a destination directory.

```
OSStatus FSCopyObjectAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this copy operation.

source

A pointer to the source object to copy. The object can be a file or a directory.

destDir

A pointer to the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags

One or more file operation option flags. See "File Operation Options" (page 303).

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval

The minimum time in seconds between callbacks within a single stage of an operation.

clientContext

User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSCopyObjectSync

Copies a source object to a destination directory.

```
OSStatus FSCopyObjectSync (
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   FSRef *target,
   OptionBits options
);
```

Parameters

source

A pointer to the source object to copy. The object can be a file or a directory.

destDir

A pointer to the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

target

A pointer to an `FSRef` variable that, on output, refers to the new object in the destination directory. This parameter is optional; pass `NULL` if you don't need to refer to the new object.

options

One or more file operation option flags. See "File Operation Options" (page 303).

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function could take a significant amount of time to execute. To avoid blocking your user interface, you should either call this function in a thread other than the main thread or use `FSCopyObjectAsync` (page 50) instead.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

FSCopyURLForVolume

Returns a copy of the URL for a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSCopyURLForVolume (
    FSVolumeRefNum vRefNum,
    CFURLRef *url
);
```

Parameters

vRefNum

The volume reference number of the target volume.

url

A pointer to a `CFURLRef` variable allocated by the caller. On return, a Core Foundation URL that specifies the location of the target volume. The caller is responsible for releasing the URL.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

FSCreateDirectoryUnicode

Creates a new directory (folder) with a Unicode name.

```
OSErr FSCreateDirectoryUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo,
   FSRef *newRef,
   FSSpecPtr newSpec,
   UInt32 *newDirID
);
```

Parameters

parentRef

A pointer to an `FSRef` specifying the parent directory where the new directory is to be created. See `FSRef` (page 223) for a description of the `FSRef` data type.

nameLength

The length of the new directory's Unicode name.

name

A pointer to the Unicode name of the new directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

whichInfo

A bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catalogInfo` parameter.

If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this parameter.

catalogInfo

A pointer to the `FSCatalogInfo` structure which specifies the values for the catalog information fields for the new directory. Specify which fields to set in the `whichInfo` parameter.

This parameter is optional; specify `NULL` if you do not wish to set catalog information for the new directory.

See `FSCatalogInfo` (page 212) for a description of the `FSCatalogInfo` data type.

newRef

On return, a pointer to the `FSRef` for the new directory. This parameter is optional; specify `NULL` if you do not want the `FSRef` returned.

newSpec

On return, a pointer to the `FSSpec` for the new directory. This parameter is optional; specify `NULL` if you do not want the `FSSpec` returned. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

newDirID

On return, a pointer to the directory ID of the directory. This parameter is optional; specify `NULL` if you do not want the directory ID returned.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
You may optionally set catalog information for the new directory using the `whichInfo` and `catalogInfo` parameters; this is equivalent to calling `FSSetCatalogInfo` (page 101) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catalogInfo` parameter. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `FSCreateDirectoryUnicode` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBDirCreateSync` (page 387). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode directory names, or other features added with HFS Plus.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest

FSCopyObject

SeeMyFriends

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

FSCreateFileUnicode

Creates a new file with a Unicode name.

```
OSErr FSCreateFileUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo,
   FSRef *newRef,
   FSSpecPtr newSpec
);
```

Parameters

parentRef
A pointer to an FSRef for the directory where the file is to be created. See FSRef (page 223) for a description of the FSRef data type.

nameLength
The length of the file's name.

name
A pointer to the Unicode name for the new file.

whichInfo
A bitmap specifying which catalog information fields to set for the new file. You specify the values for these fields in the catalogInfo parameter. If you do not wish to set catalog information for the new file, pass the constant kFSCatInfoNone. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this parameter.

catalogInfo
A pointer to the FSCatalogInfo structure which specifies the values of the new file's catalog information. Specify which fields to set in the whichInfo parameter.

This parameter is optional; specify NULL if you do not wish to set catalog information for the new file.

newRef
On return, a pointer to the FSRef for the new file. If you do not want the FSRef returned, specify NULL.

newSpec
On return, a pointer to the FSSpec for the new file. If you do not want the FSSpec returned, specify NULL. See FSSpec (page 226) for a description of the FSSpec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
You may optionally set catalog information for the new file using the whichInfo and catalogInfo parameters; this is equivalent to calling FSSetCatalogInfo (page 101) , or one of the corresponding parameter block functions, PBSetCatalogInfoSync (page 163) and PBSetCatalogInfoAsync (page 162) , after creating the file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catalogInfo` parameter. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `FSCreateFileUnicode` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateSync` (page 439). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
Cocoa PDE with Carbon Printing
FSCopyObject
Quartz2DBasics

Declared In
`Files.h`

FSCreateFork

Creates a named fork for a file or directory.

```
OSErr FSCreateFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName
);
```

Parameters

ref
A pointer to an `FSRef` specifying the file or directory. See `FSRef` (page 223) for a description of the `FSRef` data type.

forkNameLength
The length of the name of the new fork.

forkName
A pointer to the Unicode name of the fork.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `FSCreateFork` returns `errFSBadForkName` or `errFSNameTooLong`.

Discussion
A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

FSCreateVolumeOperation

Returns an `FSVolumeOperation` which can be used for an asynchronous volume operation.

```
OSStatus FSCreateVolumeOperation (
   FSVolumeOperation *volumeOp
);
```

Parameters

volumeOp
The new `FSVolumeOperation`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
When the operation is completed the `FSVolumeOperation` should be disposed of to free the memory associated with the operation using `FSDisposeVolumeOperation`.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
DockBrowser

Declared In
`Files.h`

FSDeleteFork

Deletes a named fork from a file or directory.

```
OSErr FSDeleteFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName
);
```

Parameters

ref
A pointer to an `FSRef` for the file or directory from which to delete the fork. See `FSRef` (page 223) for a description of the `FSRef` data type.

forkNameLength
The length of the Unicode name of the fork name.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

forkName

A pointer to the Unicode name of the fork to delete.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the named fork does not exist, the function returns `errFSForkNotFound`.

Discussion
Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSDeleteObject

Deletes a file or an empty directory.

```
OSErr FSDeleteObject (
   const FSRef *ref
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders). See `FSRef` (page 223) for a description of the `FSRef` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioContextInsert
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
Quartz2DBasics

Declared In
`Files.h`

FSDisposeVolumeOperation

Releases the memory associated with a volume operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSDisposeVolumeOperation (
   FSVolumeOperation volumeOp
);
```

Parameters

volumeOp

The `FSVolumeOperation` to release.

Return Value

A result code. See "File Manager Result Codes" (page 329). This function will return `paramErr` if the `FSVolumeOperation` is in use.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

DockBrowser

Declared In

`Files.h`

FSEjectVolumeAsync

Asynchronously ejects a volume.

```
OSStatus FSEjectVolumeAsync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   FSVolumeOperation volumeOp,
   void *clientData,
   FSVolumeEjectUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

vRefNum

The volume reference number of the volume to eject.

flags

Options for future use.

volumeOp

An `FSVolumeOperation` returned by `FSCreateVolumeOperation`.

clientData

A pointer to client data associated with the operation. This parameter can be `NULL`.

callback

The function to call when eject is complete.

runloop

The runloop to run on.

runloopMode

The mode for the runloop.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function starts the process of ejecting the volume specified by the *vRefNum* parameter. If a callback function is provided, that function will be called when the eject operation is complete. Once this function returns `noErr` the status of the operation can be found using `FSGetAsyncEjectStatus`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSEjectVolumeSync

Ejects a volume.

```
OSStatus FSEjectVolumeSync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   pid_t *dissenter
);
```

Parameters

vRefNum
The volume reference number of the volume to eject.

flags
Options for future use.

dissenter
On return, a pointer to the pid of the process which denied the unmount if the eject is denied.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function ejects the volume specified by the *vRefNum* parameter. If the volume cannot be ejected the pid of the process which denied the unmount will be returned in the *dissenter* parameter. This function returns after the eject is complete. Ejecting a volume will result in the unmounting of other volumes on the same device.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
AudioCDSample

Declared In
`Files.h`

FSExchangeObjects

Swaps the contents of two files.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSExchangeObjects (
   const FSRef *ref,
   const FSRef *destRef
);
```

Parameters

ref

A pointer to an `FSRef` for the first file. See `FSRef` (page 223) for a description of the `FSRef` data type.

destRef

A pointer to an `FSRef` for the second file.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSExchangeObjects` function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, `FSSpec`, or `FSRef` that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSFileOperationCancel

Cancels an asynchronous file operation.

```
OSStatus FSFileOperationCancel (
   FSFileOperationRef fileOp
);
```

Parameters

fileOp

The file operation to cancel.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function makes the specified file operation ineligible to run on any run loop. You may call this function at any time during the operation. Typically, you would use this function if the user cancels the operation. Note that to release your file operation object, you still need to call `CFRelease`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSFileOperationCopyStatus

Gets a copy of the current status information for an asynchronous file operation.

```
OSStatus FSFileOperationCopyStatus (
   FSFileOperationRef fileOp,
   FSRef *currentItem,
   FSFileOperationStage *stage,
   OSStatus *error,
   CFDictionaryRef *statusDictionary,
   void **info
);
```

Parameters

fileOp

The file operation to access.

currentItem

A pointer to an `FSRef` variable. On output, the variable contains the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

stage

A pointer to a file operation stage variable. On output, the variable contains the current stage of the file operation.

error

A pointer to an error status variable. On output, the variable contains the current error status of the file operation.

statusDictionary

A pointer to a dictionary variable. On output, the variable contains a dictionary with more detailed status information. For information about the contents of the dictionary, see "`File Operation Status Dictionary Keys`" (page 305). You should release the dictionary when you are finished using it.

info

A pointer to a generic pointer. On output, the generic pointer refers to user-defined data associated with this file operation.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

FSFileOperationCreate

Creates an object that represents an asynchronous file operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
FSFileOperationRef FSFileOperationCreate (
   CFAllocatorRef alloc
);
```

Parameters

alloc

The allocator to use. Pass `NULL` for the default allocator.

Return Value

A new FSFileOperation object, or `NULL` if the object could not be created. When you no longer need the object, you should release it by calling `CFRelease`.

Discussion

Before passing a file operation object to a function that starts an asynchronous copy or move operation, you should schedule the file operation using the function `FSFileOperationScheduleWithRunLoop` (page 62).

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

FSFileOperationGetTypeID

Returns the Core Foundation type identifier for the FSFileOperation opaque type.

```
CFTypeID FSFileOperationGetTypeID (
   void
);
```

Return Value

The type identifier for the FSFileOperation opaque type. For information about this type, see `FSFileOperationRef` (page 216).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSFileOperationScheduleWithRunLoop

Schedules an asynchronous file operation with the specified run loop and mode.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSFileOperationScheduleWithRunLoop (
   FSFileOperationRef fileOp,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

fileOp

The file operation to schedule.

runLoop

The run loop in which to schedule the operation. For information about Core Foundation run loops, see *Run Loops*.

runLoopMode

The run loop mode in which to schedule the operation. In most cases, you may specify `kCFRunLoopCommonModes`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
To run, a file operation must be scheduled with at least one run loop. A file operation can be scheduled with multiple run loop and mode combinations.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
FSFileOperation

Declared In
`Files.h`

FSFileOperationUnscheduleFromRunLoop

Unschedules an asynchronous file operation from the specified run loop and mode.

```
OSStatus FSFileOperationUnscheduleFromRunLoop (
   FSFileOperationRef fileOp,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode
);
```

Parameters

fileOp

The file operation to unschedule.

runLoop

The run loop on which to unschedule the operation.

runLoopMode

The run loop mode in which to unschedule the operation.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

FSFlushFork

Causes all data written to an open fork to be written to disk.

```
OSErr FSFlushFork (
   FSIORefNum forkRefNum
);
```

Parameters

forkRefNum

The reference number of the fork to flush.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSFlushFork` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `FSFlushFork` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `FSFlushFork` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSFlushVolume

For the specified volume, writes all open and modified files in the current process to permanent storage.

```
OSStatus FSFlushVolume (
    FSVolumeRefNum vRefNum
);
```

Parameters

vRefNum

The volume reference number of the volume to flush.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Files.h`

FSGetAsyncEjectStatus

Returns the current status of an asynchronous eject operation.

```
OSStatus FSGetAsyncEjectStatus (
   FSVolumeOperation volumeOp,
   FSEjectStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *volumeRefNum,
   pid_t *dissenter,
   void **clientData
);
```

Parameters

volumeOp
The asynchronous volume operation to get status about.

status
On return, a pointer to the status of the operation.

volumeOpStatus
If the *status* parameter is `kAsyncEjectComplete` then this contains the result code (`OSStatus`) for the operation on return.

volumeRefNum
On return, the volume reference number of the volume being ejected.

dissenter
On return, a pointer to the pid of the process which denied the unmount if the eject is denied.

clientData
On return, a pointer to client data associated with the original `FSMountServerVolumeAsync` operation.

Return Value
A result code. See "File Manager Result Codes" (page 329). A return value of `noErr` signifies that the `status` parameter has been filled with valid information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSGetAsyncMountStatus

Returns the current status of an asynchronous mount operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSGetAsyncMountStatus (
   FSVolumeOperation volumeOp,
   FSMountStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *mountedVolumeRefNum,
   void **clientData
);
```

Parameters

volumeOp

The asynchronous volume operation to get status about.

status

On return, a pointer to the status of the operation.

volumeOpStatus

If the status is `kAsyncMountComplete` then this parameter contains the result code for the operation on return.

mountedVolumeRefNum

If the status is `kAsyncMountComplete` and the *volumeOpStatus* parameter is `noErr` then this is the volume reference number for the newly mounted volume, on return.

clientData

On return, a pointer to client data associated with the original `FSMountServerVolumeAsync` operation.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

A return value of `noErr` signifies that the *status* parameter has been filled with valid information. If the status is `kAsyncMountComplete` then the rest of data returned is valid. If the status is anything else then the *volumeOpStatus* and *mountedVolumeRefNum* parameters are invalid, but the *clientData* parameter is valid.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSGetAsyncUnmountStatus

Returns the current status of an asynchronous unmount operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSGetAsyncUnmountStatus (
   FSVolumeOperation volumeOp,
   FSUnmountStatus *status,
   OSStatus *volumeOpStatus,
   FSVolumeRefNum *volumeRefNum,
   pid_t *dissenter,
   void **clientData
);
```

Parameters

volumeOp

The asynchronous volume operation to get status about.

status

On return, a pointer to the status of the operation.

volumeOpStatus

If the status is `kAsyncUnmountComplete` then this parameter contains a pointer to the result code (`OSStatus`) for the operation on return.

volumeRefNum

On return, a pointer to the volume reference number of the volume being unmounted.

dissenter

On return, a pointer to the pid of the process which denied the unmount if the unmount is denied.

clientData

On return, a pointer to client data associated with the original `FSMountServerVolumeAsync` operation.

Return Value

A result code. See "File Manager Result Codes" (page 329). A return value of `noErr` signifies that the *status* parameter has been filled with valid information.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSGetCatalogInfo

Returns catalog information about a file or directory. You can use this function to map an `FSRef` to an `FSSpec`.

```
OSErr FSGetCatalogInfo (
   const FSRef *ref,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfo,
   HFSUniStr255 *outName,
   FSSpecPtr fsSpec,
   FSRef *parentRef
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory for which to retrieve information. See `FSRef` (page 223) for a description of the `FSRef` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

whichInfo

A bitmap specifying the catalog information fields to return. If you don't want any catalog information, set `whichInfo` to the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this parameter.

catalogInfo

On return, a pointer to a catalog information structure containing the information about the file or directory. Only the information specified in the `whichInfo` parameter is returned. If you don't want any catalog information, pass `NULL` here. See `FSCatalogInfo` (page 212) for a description of the `FSCatalogInfo` data type.

outName

On return, a pointer to the Unicode name of the file or directory is returned here. This parameter is optional; if you do not wish the name returned, pass `NULL` here. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

fsSpec

On return, a pointer to the `FSSpec` for the file or directory. This parameter is optional; if you do not wish the `FSSpec` returned, pass `NULL` here. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

parentRef

On return, a pointer to the `FSRef` for the object's parent directory. This parameter is optional; if you do not wish the parent directory returned, pass `NULL` here.

If the object specified in the `ref` parameter is a volume's root directory, then the `FSRef` returned here will not be a valid `FSRef`, since the root directory has no parent object.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject
FSMegaInfo
MFSLives
ThreadsExporter
ThreadsImportMovie

Declared In
`Files.h`

FSGetCatalogInfoBulk

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSGetCatalogInfoBulk (
   FSIterator iterator,
   ItemCount maximumObjects,
   ItemCount *actualObjects,
   Boolean *containerChanged,
   FSCatalogInfoBitmap whichInfo,
   FSCatalogInfo *catalogInfos,
   FSRef *refs,
   FSSpecPtr specs,
   HFSUniStr255 *names
);
```

Parameters

iterator

The iterator to use. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88), or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, the iterator must be created with the `kFSIterateFlat` option. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

maximumObjects

The maximum number of items to return for this call.

actualObjects

On return, a pointer to the actual number of items found for this call.

containerChanged

On return, a pointer to a value indicating whether or not the container's contents have changed since the previous `FSGetCatalogInfoBulk` call. If `true`, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

This parameter is optional if you don't want this information returned, pass a `NULL` pointer.

In Mac OS X version 10.2 and later, this parameter is always set to `false`. To find out whether the container has changed since the last call to `FSGetCatalogInfoBulk`, check the modification date of the container.

whichInfo

A bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this parameter. For a description of the bits in this parameter, see "Catalog Information Bitmap Constants" (page 277).

catalogInfos

A pointer to an array of catalog information structures; one for each returned item. On input, the `catalogInfos` parameter should point to an array of `maximumObjects` catalog information structures.

This parameter is optional; if you do not wish any catalog information returned, pass `NULL` here.

refs

A pointer to an array of `FSRef` structures; one for each returned item. On input, this parameter should to point to an array of `maximumObjectsFSRef` structures.

This parameter is optional; if you do not wish any `FSRef` structures returned, pass `NULL` here.

specs

A pointer to an array of `FSSpec` structures; one for each returned item. On input, this parameter should to point to an array of `maximumObjectsFSSpec` structures.

This parameter is optional; if you do not wish any `FSSpec` structures returned, pass `NULL` here.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

names

A pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this parameter to point to an array of `maximumObjectsHFSUniStr255` structures. Otherwise, set it to `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 329). When all of the iterator's objects have been returned, the call will return `errFSNoMoreItems`.

Discussion
The `FSGetCatalogInfoBulk` call may complete and return `noErr` with fewer than `maximumObjects` items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make `FSGetCatalogInfoBulk` calls using the same iterator.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject
FSMegaInfo
MFSLives
MoreSCF

Declared In
`Files.h`

FSGetDataForkName

Returns a Unicode string constant for the name of the data fork.

```
OSErr FSGetDataForkName (
   HFSUniStr255 *dataForkName
);
```

Parameters

dataForkName

On input, a pointer to an `HFSUniStr255` structure. On return, this structure contains the Unicode name of the data fork. Currently, this is the empty string. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
There is no parameter block-based form of this call since it is not dispatched to individual volume formats, and does not require any I/O.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioCDSample

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSGetForkCBInfo

Returns information about a specified open fork, or about all open forks.

```
OSErr FSGetForkCBInfo (
   FSIORefNum desiredRefNum,
   FSVolumeRefNum volume,
   short *iterator,
   FSIORefNum *actualRefNum,
   FSForkInfo *forkInfo,
   FSRef *ref,
   HFSUniStr255 *outForkName
);
```

Parameters

desiredRefNum

If you want information on a specific fork, set this parameter to that fork's reference number, and pass `NULL` in the `iterator` parameter. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number.

Pass zero in this parameter to iterate over all open forks. You can limit this iteration to a specific volume with the `volume` parameter.

volume

The volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this parameter. To iterate over all open forks on all volumes, set this parameter to the constant `kFSInvalidVolumeRefNum`.

This parameter is ignored if you specify a fork reference number in the `desiredRefNum` parameter. Set `desiredRefNum` to zero if you wish to iterate over multiple forks.

See `FSVolumeRefNum` (page 233) for a description of the `FSVolumeRefNum` data type.

iterator

A pointer to an iterator. If the `desiredRefNum` parameter is 0, the iterator maintains state between calls to `FSGetForkCBInfo`. Set the `iterator` parameter to 0 before you begin iterating, on the first call to `FSGetForkCBInfo`. On return, the iterator will be updated; pass this updated iterator in the `iterator` parameter of the next call to `FSIterateForks` to continue iterating.

actualRefNum

On return, a pointer to the reference number of the open fork. This parameter is optional if you do not wish to retrieve the fork's reference number, pass `NULL`.

forkInfo

On return, a pointer to an `FSForkInfo` structure containing information about the open fork. This parameter is optional; if you do not wish this information returned, set `forkInfo` to `NULL`. See `FSForkInfo` (page 218) for a description of the `FSForkInfo` data type.

On OS X, the value returned by `FSGetForkCBInfo` in the `physicalEOF` field of the `FSForkInfo` structure may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `FSGetForkCBInfo` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

ref

On return, a pointer to the `FSRef` for the file or directory that contains the fork. This parameter is optional; if you do not wish to retrieve the `FSRef`, set `ref` to `NULL`. See `FSRef` (page 223) for a description of the `FSRef` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

outForkName

On return, a pointer to the name of the fork. This parameter is optional; if you do not wish the name returned, set `outForkName` to `NULL`. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If you are iterating over multiple forks, the function returns `errFSNoMoreItems` if there are no more open forks to return.

Discussion
Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use `FSGetForkCBInfo`, or one of the related parameter block functions, `PBGetForkCBInfoSync` (page 142) and `PBGetForkCBInfoAsync` (page 140) , to access information about a fork control block.

Special Considerations

Returning the fork information in the `forkInfo` parameter generally does not require a disk access; returning the information in the `ref` or `forkName` parameters may cause disk access for some volume formats.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSGetForkPosition

Returns the current position of an open fork.

```
OSErr FSGetForkPosition (
   FSIORefNum forkRefNum,
   SInt64 *position
);
```

Parameters

forkRefNum

The reference number of a fork previously opened by the `FSOpenFork` (page 87) function or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

position

On return, a pointer to the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

Return Value
A result code. See "File Manager Result Codes" (page 329).

Special Considerations

Before calling the `FSGetForkPosition` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `FSGetForkPosition` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosSync` (page 436); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

FSGetForkSize

Returns the size of an open fork.

```
OSErr FSGetForkSize (
   FSIORefNum forkRefNum,
   SInt64 *forkSize
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain this fork reference number with the FSOpenFork (page 87) function, or one of the corresponding parameter block calls, PBOpenForkSync (page 155) and PBOpenForkAsync (page 154).

forkSize

On return, a pointer to the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Special Considerations

To determine whether the FSGetForkSize function is present, call the Gestalt function. If FSGetForkSize is present, but is not directly supported by a volume, the File Manager will call PBGetEOFSync (page 430); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioCDSample
CoreTextTest

Declared In
Files.h

FSGetResourceForkName

Returns a Unicode string constant for the name of the resource fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSGetResourceForkName (
   HFSUniStr255 *resourceForkName
);
```

Parameters

resourceForkName

On input, a pointer to an `HFSUniStr255` structure. On return, this structure contains the Unicode name of the resource fork. Currently, this is "RESOURCE_FORK". See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

There is no parameter block-based form of this call since it is not dispatched to individual volume formats, and does not require any I/O.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSGetVolumeInfo

Returns information about a volume.

```
OSErr FSGetVolumeInfo (
   FSVolumeRefNum volume,
   ItemCount volumeIndex,
   FSVolumeRefNum *actualVolume,
   FSVolumeInfoBitmap whichInfo,
   FSVolumeInfo *info,
   HFSUniStr255 *volumeName,
   FSRef *rootDirectory
);
```

Parameters

volume

If you wish to obtain information on a particular volume, pass that volume's reference number here. If you wish to index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum` in this parameter. See `FSVolumeRefNum` (page 233) for a description of the `FSVolumeRefNum` data type.

volumeIndex

The index of the desired volume, or 0 to use the volume reference number in the `volume` parameter.

actualVolume

On return, a pointer to the volume reference number of the volume. This is useful when indexing over all mounted volumes. If you don't want this information (if, for instance, you supplied a particular volume reference number in the `volume`) parameter, set `actualVolume` to `NULL`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

whichInfo

A bitmap specifying which volume information fields to get and return in the `info` parameter. If you don't want information about the volume returned in the `info` parameter, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this parameter.

info

On return, a pointer to the volume information. If you don't want this output, set this parameter to `NULL`. See `FSVolumeInfo` (page 228) for a description of the `FSVolumeInfo` data type.

volumeName

On return, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

rootDirectory

On return, a pointer to the `FSRef` for the volume's root directory. If you do not wish the root directory returned, pass `NULL`. See `FSRef` (page 223) for a description of the `FSRef` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `volume` parameter and set the `volumeIndex` parameter to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the volume parameter and set `volumeIndex` to the index, starting at 1 with the first call to `FSGetVolumeInfo`.

When indexing through the list of mounted volumes, you may encounter an error with a particular volume. The terminating error code for full traversal of this list is `nsvErr`. In order to completely traverse the entire list, you may have to bump the index count when encountering other errors (for example, `ioErr`).

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 67) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 139) and `PBGetCatalogInfoAsync` (page 135).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `FSGetVolumeInfo` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioCDSample

FSMegaInfo

VolumeToBSDNode

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSGetVolumeMountInfo

Retrieves the mounting information associated with the specified volume.

```
OSStatus FSGetVolumeMountInfo (
   FSVolumeRefNum volume,
   BytePtr buffer,
   ByteCount bufferSize,
   ByteCount *actualSize
);
```

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

FSGetVolumeMountInfoSize

Determines the size of the mounting information associated with the specified volume.

```
OSStatus FSGetVolumeMountInfoSize (
   FSVolumeRefNum volume,
   ByteCount *size
);
```

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

FSGetVolumeParms

Retrieves information about the characteristics of a volume.

```
OSStatus FSGetVolumeParms (
   FSVolumeRefNum volume,
   GetVolParmsInfoBuffer *buffer,
   ByteCount bufferSize
);
```

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
FSMegaInfo

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

FSIterateForks

Determines the name and size of every named fork belonging to a file or directory.

```
OSErr FSIterateForks (
   const FSRef *ref,
   CatPositionRec *forkIterator,
   HFSUniStr255 *forkName,
   SInt64 *forkSize,
   UInt64 *forkPhysicalSize
);
```

Parameters

ref

A pointer to an FSRef specifying the file or directory to iterate. See FSRef (page 223) for a description of the FSRef data type.

forkIterator

A pointer to a structure which maintains state between calls to FSIterateForks. Before the first call, set the initialize field of the structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See CatPositionRec (page 187) for a description of the CatPositionRec data type.

forkName

On return, a pointer to the Unicode name of the fork. This parameter is optional; if you do not wish the name returned, pass a NULL pointer. See HFSUniStr255 (page 241) for a description of the HFSUniStr255 data type.

forkSize

On return, a pointer to the logical size of the fork, in bytes. This parameter is optional; if you do not wish to retrieve the logical fork size, pass a NULL pointer.

forkPhysicalSize

On return, a pointer to the physical size of the fork (that is, to the amount of space allocated on disk), in bytes. This parameter is optional; if you do not wish to retrieve the physical fork size, pass a NULL pointer.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSCopyObject

Declared In
Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSLockRange

Locks a range of bytes of the specified fork.

```
OSStatus FSLockRange (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    UInt64 requestCount,
    UInt64 *rangeStart
);
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

FSMakeFSRefUnicode

Constructs an `FSRef` for a file or directory, given a parent directory and a Unicode name.

```
OSErr FSMakeFSRefUnicode (
   const FSRef *parentRef,
   UniCharCount nameLength,
   const UniChar *name,
   TextEncoding textEncodingHint,
   FSRef *newRef
);
```

Parameters

parentRef

A pointer to the `FSRef` of the parent directory of the file or directory for which to create a new `FSRef`. See `FSRef` (page 223) for a description of the `FSRef` data type.

nameLength

The length of the file or directory name.

name

A pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to `FSMakeFSRefUnicode`.

textEncodingHint

The suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

newRef

On return, if the function returns a result of `noErr`, a pointer to the new `FSRef`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
BSDLLCTest
Cocoa PDE with Carbon Printing
FSCopyObject
Quartz2DBasics
SeeMyFriends

Declared In
`Files.h`

FSMountLocalVolumeAsync

Mounts a volume asynchronously.

```
OSStatus FSMountLocalVolumeAsync (
   CFStringRef diskID,
   CFURLRef mountDir,
   FSVolumeOperation volumeOp,
   void *clientData,
   OptionBits flags,
   FSVolumeMountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

diskID
The disk to mount.

mountDir
Pass in `NULL` ; currently only `NULL` is supported.

volumeOp
An `FSVolumeOperation` returned by `FSCreateVolumeOperation`

clientData
A pointer to client data associated with the operation. This parameter can be `NULL`.

flags
Options for future use.

callback
The function to call when mount is complete. This parameter can be `NULL`.

runloop
The runloop to run on.

runloopMode
The mode for the runloop.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function starts the process to mount the disk specified by the *diskID* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. If a callback function is provided, that function will be called when the mount operation is complete. Once this function returns `noErr` the status of the operation can be found using the `FSGetAsyncMountStatus` function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSMountLocalVolumeSync

Mounts a volume.

```
OSStatus FSMountLocalVolumeSync (
   CFStringRef diskID,
   CFURLRef mountDir,
   FSVolumeRefNum *mountedVolumeRefNum,
   OptionBits flags
);
```

Parameters

diskID
The disk to mount.

mountDir
Pass in `NULL`; currently only `NULL` is supported.

mountedVolumeRefNum
On return, a pointer to the volume reference number of the newly mounted volume.

flags
Options for future use.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function mounts the disk specified by the *diskID* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. This function returns after the mount is complete.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSMountServerVolumeAsync

Mounts a server volume asynchronously.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSMountServerVolumeAsync (
   CFURLRef url,
   CFURLRef mountDir,
   CFStringRef user,
   CFStringRef password,
   FSVolumeOperation volumeOp,
   void *clientData,
   OptionBits flags,
   FSVolumeMountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

url

The server to mount.

mountDir

The directory to mount the server to. If this parameter is `NULL`, the default location is used.

user

A string to pass as the user for authentication. This parameter can be `NULL`.

password

A string to pass as the password for authenticated log in. This parameter can be `NULL`.

volumeOp

An `FSVolumeOperation` returned by the `FSCreateVolumeOperation` function.

clientData

A pointer to client data associated with the operation. This parameter can be `NULL`.

flags

Options for future use.

callback

A function to call when the mount is complete. This parameter can be `NULL`.

runloop

The runloop to run on.

runloopMode

The mode for the runloop.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function will start the process to mount the server specified by the *url* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. An optional user and password can be passed in for authentication. If no user or password is provided then the underlying file system will handle authentication if required. If a callback function is provided, that function will be called when the mount operation is complete. Once this function returns `noErr` the status of the operation can be found using the `FSGetAsyncMountStatus` function.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

DockBrowser

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

FSMountServerVolumeSync

Mounts a server volume.

```
OSStatus FSMountServerVolumeSync (
   CFURLRef url,
   CFURLRef mountDir,
   CFStringRef user,
   CFStringRef password,
   FSVolumeRefNum *mountedVolumeRefNum,
   OptionBits flags
);
```

Parameters

url
The server to mount.

mountDir
The directory to mount the server to. If this parameter is `NULL`, the default location is used.

user
A string to pass as the user for authentication.

password
A string to pass as the password for authenticated log in.

mountedVolumeRefNum
On return, a pointer to the volume reference number of the newly mounted volume.

flags
Options for future use.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function will mount the server specified by the *url* parameter at the location specified by the *mountDir* parameter. If *mountDir* is `NULL`, the default location is used. An optional user and password can be passed in for authentication. If no user or password is provided then the underlying file system will handle authentication if required. This function returns after the mount is complete.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSMoveObject

Moves a file or directory into a different directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSMoveObject (
   const FSRef *ref,
   const FSRef *destDirectory,
   FSRef *newRef
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory to move. See `FSRef` (page 223) for a description of the `FSRef` data type.

destDirectory

A pointer to an `FSRef` specifying the directory into which the file or directory indicated by the `ref` parameter will be moved.

newRef

On return, a pointer to the new `FSRef` for the file or directory in its new location. This parameter is optional; if you do not wish the `FSRef` returned, pass `NULL`.

Return Value

A result code. See "File Manager Result Codes" (page 329). If the `destDirectory` parameter specifies a non-existent object, `dirNFErr` is returned; if it refers to a file, `errFSNotAFolder` is returned. If the directory specified in the `destDirectory` parameter is on a different volume than the file or directory indicated in the `ref` parameter, `diffVolErr` is returned.

Discussion

Moving an object may change its `FSRef`. If you want to continue to refer to the object, you should pass a non- `NULL` pointer in the `newRef` parameter and use the `FSRef` returned there to refer to the object after the move. The original `FSRef` passed in the `ref` parameter may or may not be usable after the move. The `newRef` parameter may point to the same storage as the `destDirectory` or `ref` parameters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSMoveObjectAsync

Starts an asynchronous file operation to move a source object to a destination directory.

```
OSStatus FSMoveObjectAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this move operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

source

A pointer to the source object to move. The object can be a file or a directory.

destDir

A pointer to the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags

One or more file operation option flags. See "File Operation Options" (page 303). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval

The minimum time in seconds between callbacks within a single stage of an operation.

clientContext

User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
FSFileOperation

Declared In
`Files.h`

FSMoveObjectSync

Moves a source object to a destination directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSMoveObjectSync (
   const FSRef *source,
   const FSRef *destDir,
   CFStringRef destName,
   FSRef *target,
   OptionBits options
);
```

Parameters

source

A pointer to the source object to move. The object can be a file or a directory. On output, the source object is no longer valid; if you want to refer to the moved object, you should use the `FSRef` variable passed back in the `target` parameter.

destDir

A pointer to the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

target

A pointer to an `FSRef` variable that, on output, refers to the new object in the destination directory. This parameter is optional; pass `NULL` if you don't need to refer to the new object.

options

One or more file operation option flags. See "File Operation Options" (page 303). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If the destination directory is on the same volume as the source object, this is a fast operation. If the move is across volumes, this function could take a significant amount of time to execute; you should either call it in a thread other than the main thread or use `FSMoveObjectAsync` (page 83) instead.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

FSMoveObjectToTrashAsync

Starts an asynchronous file operation to move a source object to the Trash.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSMoveObjectToTrashAsync (
   FSFileOperationRef fileOp,
   const FSRef *source,
   OptionBits flags,
   FSFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this move operation. For more information, see the function `FSFileOperationCreate` (page 61).

source

A pointer to the source object to move. The object can be a file or a directory.

flags

One or more file operation option flags. See "File Operation Options" (page 303).

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval

The minimum time in seconds between callbacks within a single stage of an operation.

clientContext

User-defined data to associate with this operation. This data is passed to the function you specify in the *callback* parameter. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function starts an asynchronous file operation to move the object specified by the *source* parameter to the Trash. If the source volume does not support a trash folder, the operation will fail and return an error to the status callback specified in the *callback* parameter. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Status callbacks occur on one of the runloop and mode combinations on which the operation was scheduled. Upon successful completion of the operation, the last *currentItem* parameter (passed to the last status callback or retrieved by calling `FSFileOperationCopyStatus` (page 61)) is the object in the Trash.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`Files.h`

FSMoveObjectToTrashSync

Moves a source object to the Trash.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSMoveObjectToTrashSync (
   const FSRef *source,
   FSRef *target,
   OptionBits options
);
```

Parameters

source

A pointer to the source object to move. The object can be a file or a directory. On output, the source object is no longer valid; if you want to refer to the moved object, you should use the value passed back in the *target* parameter.

target

A pointer to the target object that, on output, resides in a trash folder. This parameter is optional; pass `NULL` if you don't need to refer to this object.

options

One or more file operation option flags. See "File Operation Options" (page 303).

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function moves a file or directory to the Trash, adjusting the object's name if necessary. The appropriate trash folder is chosen based on the source volume and the current user. If the source volume does not support a trash folder, this function does nothing and returns an error. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Files.h`

FSOpenFork

Opens any fork of a file or directory for streaming access.

```
OSErr FSOpenFork (
   const FSRef *ref,
   UniCharCount forkNameLength,
   const UniChar *forkName,
   SInt8 permissions,
   FSIORefNum *forkRefNum
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory owning the fork to open. See `FSRef` (page 223) for a description of the `FSRef` data type.

forkNameLength

The length of the fork name in Unicode characters.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

forkName

A pointer to the Unicode name of the fork to open. You can obtain the string constants for the data fork and resource fork names using the `FSGetDataForkName` (page 70) and `FSGetResourceForkName` (page 73) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

permissions

A constant indicating the type of access which you wish to have to the fork via the returned fork reference. This parameter is the same as the `permission` parameter passed to the `FSpOpenDF` and `FSpOpenRF` functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 295).

forkRefNum

On return, a pointer to the fork reference number for accessing the open fork.

Return Value

A result code. See "File Manager Result Codes" (page 329). On some file systems, `FSOpenFork` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion

When you use this function to open a file on a local volume and pass in a permissions value of `fsCurPerm`, `fsWrPerm`, or `fsRdWrPerm`, Mac OS X does not guarantee exclusive file access. Before making any assumptions about the underlying file access, you should always check to see whether the Supports Exclusive Locks feature is available. If this feature is not available, your application cannot know whether another application has access to the same file. For more information, see ADC Technical Note TN2037.

To access named forks or forks larger than 2GB, you must use the `FSOpenFork` function or one of the corresponding parameter block calls: `PBOpenForkSync` and `PBOpenForkAsync`. To determine if the `FSOpenFork` function is present, call the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioCDSample

Cocoa PDE with Carbon Printing

CoreTextTest

FSCopyObject

Declared In

`Files.h`

FSOpenIterator

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSOpenIterator (
   const FSRef *container,
   FSIteratorFlags iteratorFlags,
   FSIterator *iterator
);
```

Parameters

container

A pointer to an FSRef for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate). See FSRef (page 223) for a description of the FSRef data type.

iteratorFlags

A set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 310) for a description of the flags defined for this parameter.

Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the kFSIterateSubtree flag in this parameter returns errFSBadIteratorFlags. To determine whether subtree iterators are supported, check that the bSupportsSubtreeIterators bit returned by PBHGetVolParmsSync (page 518) or PBHGetVolParmsAsync (page 517) is set.

iterator

On return, a pointer to the new FSIterator. You can pass this iterator to the FSGetCatalogInfoBulk (page 68) or FSCatalogSearch (page 45) functions and their parameter block-based counterparts.

The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

See FSIterator (page 221) for a description of the FSIterator data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the FSCloseIterator (page 48) function, or one of the related parameter block functions, PBCloseIteratorSync (page 119) and PBCloseIteratorAsync (page 119) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the Gestalt function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

FSCopyObject

FSMegaInfo

MFSLives

MoreSCF

Declared In

Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSPathCopyObjectAsync

Starts an asynchronous file operation to copy a source object to a destination directory using pathnames.

```
OSStatus FSPathCopyObjectAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this copy operation.

sourcePath

The UTF-8 pathname of the source object to copy. The object can be a file or a directory.

destDirPath

The UTF-8 pathname of the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags

One or more file operation option flags. See "File Operation Options" (page 303).

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval

The minimum time in seconds between callbacks within a single stage of an operation.

clientContext

User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSPathCopyObjectSync

Copies a source object to a destination directory using pathnames.

```
OSStatus FSPathCopyObjectSync (
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath

The UTF-8 pathname of the source object to copy. The object can be a file or a directory.

destDirPath

The UTF-8 pathname of the destination directory.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

targetPath

A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the new object in the destination directory. If the operation fails, the pathname is set to `NULL`. When you no longer need the pathname, you should free it. This parameter is optional; pass `NULL` if you don't need the pathname.

options

One or more file operation option flags. See "File Operation Options" (page 303).

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function could take a significant amount of time to execute. To avoid blocking your user interface, you should either call this function in a thread other than the main thread or use `FSPathCopyObjectAsync` (page 90) instead.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

`Files.h`

FSPathFileOperationCopyStatus

Gets a copy of the current status information for an asynchronous file operation that uses pathnames.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSPathFileOperationCopyStatus (
   FSFileOperationRef fileOp,
   char **currentItem,
   FSFileOperationStage *stage,
   OSStatus *error,
   CFDictionaryRef *statusDictionary,
   void **info
);
```

Parameters

fileOp

The file operation to access.

currentItem

A pointer to a char* variable. On output, the variable refers to the UTF-8 pathname of the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory). You should free the pathname when you are finished using it.

stage

A pointer to a file operation stage variable. On output, the variable contains the current stage of the file operation.

error

A pointer to an error status variable. On output, the variable contains the current error status of the file operation.

statusDictionary

A pointer to a dictionary variable. On output, the variable contains a dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 305). You should release the dictionary when you are finished using it.

info

A pointer to a generic pointer. On output, the generic pointer refers to user-defined data associated with this file operation.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

FSFileOperation

Declared In

Files.h

FSPathMakeRef

Converts a POSIX-style pathname into an `FSRef` structure.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSPathMakeRef (
    const UInt8 *path,
    FSRef *ref,
    Boolean *isDirectory
);
```

Parameters

path

A UTF-8 C string that contains the pathname to convert.

ref

A pointer to an `FSRef` structure allocated by the caller. On output, the `FSRef` structure refers to the object whose location is specified by the *path* parameter.

isDirectory

A pointer to a Boolean variable allocated by the caller. On output, `true` indicates the object is a directory. This parameter is optional and may be `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FSMegaInfo
FSReplaceObject
MFSLives
ThreadsExporter
ThreadsImportMovie

Declared In
`Files.h`

FSPathMakeRefWithOptions

Converts a POSIX-style pathname into an `FSRef` structure with options.

```
OSStatus FSPathMakeRefWithOptions (
   const UInt8 *path,
   OptionBits options,
   FSRef *ref,
   Boolean *isDirectory
);
```

Parameters

path

A UTF-8 C string that contains the pathname to convert.

options

One or more conversion flags. See "Path Conversion Options" (page 314).

ref

A pointer to an `FSRef` structure allocated by the caller. On output, the `FSRef` structure refers to the object whose location is specified by the *path* parameter. If the object is a symbolic link, the *options* parameter determines whether the `FSRef` structure refers to the link itself or to the linked object.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

isDirectory

A pointer to a Boolean variable allocated by the caller. On output, `true` indicates the object is a directory. This parameter is optional and may be `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

FSPathMoveObjectAsync

Starts an asynchronous file operation to move a source object to a destination directory using pathnames.

```
OSStatus FSPathMoveObjectAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp

The file operation object you created for this move operation.

sourcePath

The UTF-8 pathname of the source object to move. The object can be a file or a directory.

destDirPath

The UTF-8 pathname of the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

flags

One or more file operation option flags. See "File Operation Options" (page 303). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

callback

A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval

The minimum time in seconds between callbacks within a single stage of an operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

clientContext

User-defined data to associate with this operation. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If you specify a status callback function, status callbacks will occur in one of the run loop and mode combinations with which you scheduled the file operation.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
FSFileOperation

Declared In
`Files.h`

FSPathMoveObjectSync

Moves a source object to a destination directory using pathnames.

```
OSStatus FSPathMoveObjectSync (
   const char *sourcePath,
   const char *destDirPath,
   CFStringRef destName,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath

The UTF-8 pathname of the source object to move. The object can be a file or a directory.

destDirPath

The UTF-8 pathname of the destination directory. If the destination directory is not on the same volume as the source object, the source object is copied and then deleted.

destName

The name for the new object in the destination directory. Pass `NULL` to use the name of the source object.

targetPath

A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the new object in the destination directory. When you no longer need the pathname, you should free it. If the operation fails, the pathname is set to `NULL`. This parameter is optional; pass `NULL` if you don't need the pathname.

options

One or more file operation option flags. See "File Operation Options" (page 303). If you specify the `kFSFileOperationDoNotMoveAcrossVolumes` flag and the destination directory is not on the same volume as the source object, this function does nothing and returns an error.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If the destination directory is on the same volume as the source object, this is a fast operation. If the move is across volumes, this function could take a significant amount of time to execute; you should call it in a thread other than the main thread or use `FSPathMoveObjectAsync` (page 94) instead.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
FSFileOperation

Declared In
`Files.h`

FSPathMoveObjectToTrashAsync

Starts an asynchronous file operation to move a source object, specified using a pathname, to the Trash.

```
OSStatus FSPathMoveObjectToTrashAsync (
   FSFileOperationRef fileOp,
   const char *sourcePath,
   OptionBits flags,
   FSPathFileOperationStatusProcPtr callback,
   CFTimeInterval statusChangeInterval,
   FSFileOperationClientContext *clientContext
);
```

Parameters

fileOp
The file operation object you created for this move operation. For more information, see the function `FSFileOperationCreate` (page 61).

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

flags
One or more file operation option flags. See "File Operation Options" (page 303).

callback
A callback function to receive status updates as the file operation proceeds. For more information, see "File Operation Callbacks" (page 174). This parameter is optional; pass `NULL` if you don't need to supply a status callback.

statusChangeInterval
The minimum time in seconds between callbacks within a single stage of an operation.

clientContext
User-defined data to associate with this operation. This data is passed to the function you specify in the *callback* parameter. For more information, see `FSFileOperationClientContext` (page 215). This parameter is optional; pass `NULL` if you don't need to supply a client context.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
This function starts an asynchronous file operation to move the object specified by the *sourcePath* parameter to the Trash. If the source volume does not support a trash folder, the operation will fail and return an error to the status callback specified in the *callback* parameter. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Status callbacks occur on one of the runloop and mode combinations on which the operation was scheduled. Upon successful completion of the operation, the last *currentItem* parameter (passed to the last status callback or retrieved by calling `FSFileOperationCopyStatus` (page 61)) is the object in the Trash.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Files.h`

FSPathMoveObjectToTrashSync

Moves a source object, specified using a pathname, to the Trash.

```
OSStatus FSPathMoveObjectToTrashSync (
   const char *sourcePath,
   char **targetPath,
   OptionBits options
);
```

Parameters

sourcePath
The UTF-8 pathname of the source object to move. The object can be a file or a directory.

targetPath
A pointer to a char* variable that, on output, refers to the UTF-8 pathname of the target object in the Trash. When you no longer need the pathname, you should free it. If the operation fails, the pathname is set to `NULL`. This parameter is optional; pass `NULL` if you don't need the pathname.

options
One or more file operation option flags. See "File Operation Options" (page 303).

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
This function moves a file or directory to the Trash, adjusting the object's name if necessary. The appropriate trash folder is chosen based on the source volume and the current user. If the source volume does not support a trash folder, this function does nothing and returns an error. (This is the same circumstance that triggers the delete immediately behavior in the Finder.)

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSReadFork

Reads data from an open fork.

```
OSErr FSReadFork (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    ByteCount requestCount,
    void *buffer,
    ByteCount *actualCount
);
```

Parameters

forkRefNum

The reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 87) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

positionMode

A constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base location.

The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently.

If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches.

See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching the read.

positionOffset

The offset from the base location for the start of the read.

requestCount

The number of bytes to read.

buffer

A pointer to the buffer where the data will be returned.

actualCount

On return, a pointer to the number of bytes actually read. The value pointed to by the `actualCount` parameter should be equal to the value in the `requestCount` parameter unless there was an error during the read operation.

This parameter is optional; if you don't want this information returned, set `actualCount` to `NULL`.

Return Value

A result code. See "File Manager Result Codes" (page 329). If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
`FSReadFork` reads data starting at the position specified by the `positionMode` and `positionOffset` parameters. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` parameter and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` parameter and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AudioCDSample
Cocoa PDE with Carbon Printing
CoreTextTest
FSCopyObject
FSCreateFileAndOpenForkUnicode

Declared In
`Files.h`

FSRefMakePath

Converts an `FSRef` structure into a POSIX-style pathname.

```
OSStatus FSRefMakePath (
   const FSRef *ref,
   UInt8 *path,
   UInt32 maxPathSize
);
```

Parameters

ref

A pointer to the `FSRef` structure to convert.

path

A pointer to a character buffer allocated by the caller. On output, the buffer contains a UTF-8 C string that specifies the absolute path to the object referred to by the *ref* parameter. The File Manager uses the *maxPathSize* parameter to make sure it does not overrun the buffer.

maxPathSize

The maximum number of bytes to copy into the buffer.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
ColorSyncDevices-Cocoa
FileNotification
FSCopyObject
FSMegaInfo
ImageApp

Declared In
`Files.h`

FSRenameUnicode

Renames a file or folder.

```
OSErr FSRenameUnicode (
   const FSRef *ref,
   UniCharCount nameLength,
   const UniChar *name,
   TextEncoding textEncodingHint,
   FSRef *newRef
);
```

Parameters

ref

A pointer to an `FSRef` for the file or directory to rename. See `FSRef` (page 223) for a description of the `FSRef` data type.

nameLength

The length of the new name in Unicode characters.

name

A pointer to the new Unicode name of the file or directory.

textEncodingHint

The suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

newRef

On return, a pointer to the new `FSRef` for the file or directory. This parameter is optional; if you do not wish the `FSRef` returned, pass `NULL`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Because renaming an object may change its `FSRef`, you should pass a non- `NULL` pointer in the `newRef` parameter and use the `FSRef` returned there to access the object after the renaming, if you wish to continue to refer to the object. The `FSRef` passed in the `ref` parameter may or may not be usable after the object is renamed. The `FSRef` returned in the `newRef` parameter may point to the same storage as the `FSRef` passed in `ref`.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

FSSetCatalogInfo

Sets catalog information about a file or directory.

```
OSErr FSSetCatalogInfo (
   const FSRef *ref,
   FSCatalogInfoBitmap whichInfo,
   const FSCatalogInfo *catalogInfo
);
```

Parameters

ref

A pointer to an `FSRef` specifying the file or directory whose information is to be changed. See `FSRef` (page 223) for a description of the `FSRef` data type.

whichInfo

A bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` parameter. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this parameter.

To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this parameter. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `FSSetCatalogInfo` cannot set the user and group IDs, it returns an error.

catalogInfo

A pointer to the structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

- `createDate`
- `contentModDate`
- `attributeModDate`
- `accessDate`
- `backupDate`
- `permissions`
- `finderInfo`
- `extFinderInfo`
- `textEncodingHint`

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
BSDLLCTest
FSCopyObject
FSRemoveInheritedACEs
Quartz2DBasics

Declared In
`Files.h`

FSSetForkPosition

Sets the current position of an open fork.

```
OSErr FSSetForkPosition (
   FSIORefNum forkRefNum,
   UInt16 positionMode,
   SInt64 positionOffset
);
```

Parameters

forkRefNum
The reference number of a fork previously opened by the `FSOpenFork` (page 87), `PBOpenForkSync` (page 155), or `PBOpenForkAsync` (page 154) function.

positionMode
A constant specifying the base location within the fork for the new position. If this parameter is equal to `fsAtMark`, then the `positionOffset` parameter is ignored. See "Position Mode Constants" (page 314) for a description of the constants you can use to specify the base location.

positionOffset
The offset of the new position from the base location specified in the `positionMode` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 329). This function returns the result code `posErr` if you attempt to set the current position of the fork to an offset before the start of the file.

Special Considerations

To determine if the `FSSetForkPosition` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `FSSetForkPosition` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosSync` (page 486) function. However, if the volume does not directly support the `FSSetForkPosition` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSSetForkSize

Changes the size of an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSSetForkSize (
   FSIORefNum forkRefNum,
   UInt16 positionMode,
   SInt64 positionOffset
);
```

Parameters

forkRefNum

The reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

positionMode

A constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 314) for more information about the constants you can use to specify the base location.

positionOffset

The offset of the new size from the base location specified in the `positionMode` parameter.

Return Value

A result code. See "File Manager Result Codes" (page 329). If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

Discussion

The `FSSetForkSize` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` parameters. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size the current position will be equal to the logical end-of-file.

Special Considerations

You do not need to check that the volume supports the `FSSetForkSize` function. If a volume does not support the `FSSetForkSize` function, but the `FSSetForkSize` function is present, the File Manager automatically calls the `PBSetEOFSync` (page 484) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `FSSetForkSize` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `FSSetForkSize` function is present, call the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

FSCopyObject

Declared In

`Files.h`

FSSetVolumeInfo

Sets information about a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSSetVolumeInfo (
   FSVolumeRefNum volume,
   FSVolumeInfoBitmap whichInfo,
   const FSVolumeInfo *info
);
```

Parameters

volume

The volume reference number of the volume whose information is to be changed. See `FSVolumeRefNum` (page 233) for a description of the `FSVolumeRefNum` data type.

whichInfo

A bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this parameter.

info

A pointer to the new volume information. See `FSVolumeInfo` (page 228) for a description of the `FSVolumeInfo` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 101) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSUnlockRange

Unlocks a range of bytes of the specified fork.

```
OSStatus FSUnlockRange (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    UInt64 requestCount,
    UInt64 *rangeStart
);
```

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSUnmountVolumeAsync

Unmounts a volume asynchronously.

```
OSStatus FSUnmountVolumeAsync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   FSVolumeOperation volumeOp,
   void *clientData,
   FSVolumeUnmountUPP callback,
   CFRunLoopRef runloop,
   CFStringRef runloopMode
);
```

Parameters

vRefNum

The volume reference number of the volume to unmount.

flags

Options for future use.

volumeOp

An `FSVolumeOperation` returned by the `FSCreateVolumeOperation` function.

clientData

A pointer to client data associated with the operation.

callback

The function to call when the unmount is complete.

runloop

The runloop to run on.

runloopMode

The mode for the runloop.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function starts the process of unmounting the volume specified by the *vRefNum* parameter. If a callback function is provided, that function will be called when the unmount operation is complete. Once this function returns `noErr` the status of the operation can be found using the `FSGetAsyncUnmountStatus` function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

FSUnmountVolumeSync

Unmounts a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus FSUnmountVolumeSync (
   FSVolumeRefNum vRefNum,
   OptionBits flags,
   pid_t *dissenter
);
```

Parameters

vRefNum

The volume reference number of the volume to unmount.

flags

Options for future use.

dissenter

On return, a pointer to the pid of the process which denied the unmount if the unmount is denied.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

This function unmounts the volume specified by the *vRefNum* parameter. If the volume cannot be unmounted the pid of the process which denied the unmount will be returned in the *dissenter* parameter. This function returns after the unmount is complete.

Availability

Available in Mac OS X v10.2 and later.

Declared In

Files.h

FSVolumeMount

Mounts a volume using the specified mounting information.

```
OSStatus FSVolumeMount (
   BytePtr buffer,
   FSVolumeRefNum *mountedVolume
);
```

Availability

Available in Mac OS X v10.5 and later.

Declared In

Files.h

FSWriteFork

Writes data to an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSWriteFork (
    FSIORefNum forkRefNum,
    UInt16 positionMode,
    SInt64 positionOffset,
    ByteCount requestCount,
    const void *buffer,
    ByteCount *actualCount
);
```

Parameters

forkRefNum

The reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

positionMode

A constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base location.

The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching.

positionOffset

The offset from the base location for the start of the write.

requestCount

The number of bytes to write.

buffer

A pointer to a buffer containing the data to write.

actualCount

On return, a pointer to the number of bytes actually written. The value pointed to by the `actualCount` parameter will be equal to the value in the `requestCount` parameter unless there was an error during the write operation.

This parameter is optional; if you don't want this information, set `actualCount` to `NULL`.

Return Value

A result code. See "File Manager Result Codes" (page 329). If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

Discussion

`FSWriteFork` writes data starting at the position specified by the `positionMode` and `positionOffset` parameters. The function attempts to write `requestCount` bytes from the buffer pointed at by the `buffer` parameter and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

FSCopyObject

FSCreateFileAndOpenForkUnicode

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

InvokeFNSubscriptionUPP

Calls your directory change callback function.

```
void InvokeFNSubscriptionUPP (
   FNMessage message,
   OptionBits flags,
   void *refcon,
   FNSubscriptionRef subscription,
   FNSubscriptionUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the directory change function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on directory change functions, see `FNSubscriptionProcPtr` (page 174).

Availability
Available in Mac OS X v10.1 and later.

Declared In
Files.h

InvokeFSVolumeEjectUPP

Calls your volume ejection callback function.

```
void InvokeFSVolumeEjectUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum volumeRefNum,
   pid_t dissenter,
   FSVolumeEjectUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume ejection function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeEjectProcPtr` (page 177).

Availability
Available in Mac OS X v10.2 and later.

Declared In
Files.h

InvokeFSVolumeMountUPP

Calls your volume mount callback function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void InvokeFSVolumeMountUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum mountedVolumeRefNum,
   FSVolumeMountUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume mount function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeMountProcPtr` (page 178).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

InvokeFSVolumeUnmountUPP

Calls your volume unmount callback function.

```
void InvokeFSVolumeUnmountUPP (
   FSVolumeOperation volumeOp,
   void *clientData,
   OSStatus err,
   FSVolumeRefNum volumeRefNum,
   pid_t dissenter,
   FSVolumeUnmountUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the volume unmount function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on change notification functions, see `FSVolumeUnmountProcPtr` (page 179).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

InvokeIOCompletionUPP

Calls your I/O completion callback function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void InvokeIOCompletionUPP (
   ParmBlkPtr paramBlock,
   IOCompletionUPP userUPP
);
```

Discussion
The File Manager calls this function to invoke the I/O completion function which you have provided for use after an asynchronous call has been completed. You should not need to use this function yourself. For more information on I/O completion functions, see `IOCompletionProcPtr` (page 179).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

NewFNSubscriptionUPP

Creates a new universal procedure pointer (UPP) to your directory change callback function.

```
FNSubscriptionUPP NewFNSubscriptionUPP (
   FNSubscriptionProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to a directory change callback function. For more information, see `FNSubscriptionProcPtr` (page 174).

Return Value
A UPP to your directory change callback function.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

NewFSVolumeEjectUPP

Creates a new universal procedure pointer (UPP) to your volume ejection callback function.

```
FSVolumeEjectUPP NewFSVolumeEjectUPP (
   FSVolumeEjectProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to a volume ejection callback function. For more information, see `FSVolumeEjectProcPtr` (page 177).

Return Value
A UPP to your volume ejection callback function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

NewFSVolumeMountUPP

Creates a new universal procedure pointer (UPP) to your volume mount callback function.

```
FSVolumeMountUPP NewFSVolumeMountUPP (
   FSVolumeMountProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a volume mount callback function. For more information, see `FSVolumeEjectProcPtr` (page 177).

Return Value
A UPP to your volume mount callback function.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
DockBrowser

Declared In
`Files.h`

NewFSVolumeUnmountUPP

Creates a new universal procedure pointer (UPP) to your volume unmount callback function.

```
FSVolumeUnmountUPP NewFSVolumeUnmountUPP (
   FSVolumeUnmountProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a volume unmount callback function. For more information, see `FSVolumeUnmountProcPtr` (page 179).

Return Value
A UPP to your volume unmount callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

NewIOCompletionUPP

Creates a new universal procedure pointer (UPP) to your I/O completion callback function.

```
IOCompletionUPP NewIOCompletionUPP (
   IOCompletionProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your I/O completion callback function. For more information, see `IOCompletionProcPtr` (page 179).

Return Value

A UPP to your I/O completion callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBAllocateForkAsync

Allocates space on a volume to an open fork.

```
void PBAllocateForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 87) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`allocationFlags`

On input, a constant indicating how the new space should be allocated. See "Allocation Flags" (page 273) for a description of the constants which you can use in this field.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`positionMode`

On input, a constant specifying the base location within the fork for the start of the allocation. See "Position Mode Constants" (page 314) for more information on the constants which you can use to specify the base location.

`positionOffset`

On input, the offset from the base location of the start of the allocation.

`allocationAmount`

On input, the number of bytes to allocate. On output, the number of bytes actually allocated to the file. The number of bytes allocated may be smaller than the requested amount if some of the space was already allocated. The value returned in this field does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The `PBAllocateForkAsync` function attempts to allocate the number of requested bytes of physical storage starting at the offset specified by the `positionMode` and `positionOffset` fields. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using `FSCloseFork` (page 47) , `PBCloseForkSync` (page 118) , or `PBCloseForkAsync` (page 117) ; or when flushed, using `FSFlushFork` (page 64) , `PBFlushForkSync` (page 133) , or `PBFlushForkAsync` (page 133).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBAllocateForkSync

Allocates space on a volume to an open fork.

```
OSErr PBAllocateForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 87) function, or with one of the corresponding parameter block functions, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`allocationFlags`

On input, a constant indicating how the new space should be allocated. See "Allocation Flags" (page 273) for a description of the constants you can use in this field.

`positionMode`

On input, a constant specifying the base location within the fork for the start of the allocation. See "Position Mode Constants" (page 314) for more information on the constants which you can use to specify the base location.

`positionOffset`

On input, the offset from the base location of the start of the allocation.

`allocationAmount`

On input, the number of bytes to allocate. On output, the number of bytes actually allocated to the file. The number of bytes allocated may be smaller than the requested amount if some of the space was already allocated. The value returned in this field does not reflect any additional bytes that may have been allocated because space is allocated in terms of fixed units such as allocation blocks, or the use of a clump size to reduce fragmentation.

The `PBAllocateForkSync` function attempts to allocate the number of requested bytes of physical storage starting at the offset specified by the `positionMode` and `positionOffset` fields. For volume formats that support preallocated space, you can later write to this range of bytes (including extending the size of the fork) without requiring an implicit allocation.

Any extra space allocated but not used will be deallocated when the fork is closed, using `FSCloseFork` (page 47) , `PBCloseForkSync` (page 118) , or `PBCloseForkAsync` (page 117) ; or when flushed, using `FSFlushFork` (page 64) , `PBFlushForkSync` (page 133) , or `PBFlushForkAsync` (page 133).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCatalogSearchAsync

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
void PBCatalogSearchAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Discussion
The relevant fields of this parameter are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`

On output, the result code of the function. When the entire volume has been searched, `errFSNoMoreItems` is returned.

`iterator`

On input, the iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchParams` field. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 221) for more information on the `FSIterator` data type.

`searchParams`

On input, a pointer to an `FSSearchParams` (page 225) structure containing the search criteria. You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 212) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 384) and `PBCatSearchAsync` (page 382) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 286) for a description of the search bits. There are a few new search criteria supported by `PBCatalogSearchAsync` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants described in "Catalog Search Constants" (page 285).If the `searchTime` field of this structure is non-zero, it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumItems` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search. If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this field to point to `FSCatalogInfo` structures containing the values to match against.

`maximumItems`

On input, the maximum number of items to return for this call.

`actualItems`

On output, the actual number of items returned for this call.

`containerChanged`

On output, a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `PBCatalogSearchAsync` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`

On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

`catalogInfo`

On output, a pointer to an array of `FSCatalogInfo` (page 212) structures; one for each found item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here.

`refs`

On output, a pointer to an array of `FSRef` (page 223) structures; one for each returned item. On input, if you want an `FSRef` for each item found, pass a pointer to an array of `maximumItems FSRef` structures. Otherwise, pass `NULL`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`names`

On output, a pointer to an array of filenames; one for each returned item. On input, if you want the Unicode filename for each item found, pass a pointer to an array of `maximumItems` `HFSUniStr255` (page 241) structures. Otherwise, pass `NULL`.

A single search may span more than one call to `PBCatalogSearchAsync`. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (`searchTime`) has been reached or `maximumItems` items have been returned. If the search is not completed, you can continue the search by making another call to `PBCatalogSearchAsync` and passing the updated iterator returned by the previous call in the `iterator` field.

Before calling this function, you should determine that it is present, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCatalogSearchSync

Searches for objects traversed by a catalog iterator that match a given set of criteria.

```
OSErr PBCatalogSearchSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). When the entire volume has been searched, `errFSNoMoreItems` is returned.

Discussion
The relevant fields of this parameter are:

`iterator`

On input, the iterator to use. Objects traversed by this iterator are matched against the criteria specified by the `searchParams` field. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, this iterator must be created with the `kFSIterateSubtree` option and the container must be the root directory of a volume. See `FSIterator` (page 221) for more information on the `FSIterator` data type.

`searchParams`

On input, a pointer to an `FSSearchParams` (page 225) structure containing the search criteria. You can match against the object's name in Unicode and by the fields in an `FSCatalogInfo` (page 212) structure. You may use the same search bits as passed in the `ioSearchBits` field to the `PBCatSearchSync` (page 384) and `PBCatSearchAsync` (page 382) functions; they control the corresponding `FSCatalogInfo` fields. See "Catalog Search Masks" (page 286) for a description of the search bits. There are a few new search criteria supported by `PBCatalogSearchSync` but not by `PBCatSearchSync` and `PBCatSearchAsync`. These new search criteria are indicated by the constants

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

described in "Catalog Search Constants" (page 285).If the `searchTime` field of this structure is non-zero, it is interpreted as a Time Manager duration; the search may terminate after this duration even if `maximumItems` objects have not been returned and the entire catalog has not been scanned. If `searchTime` is zero, there is no time limit for the search. If you are searching by any criteria other than name, you must set the `searchInfo1` and `searchInfo2` fields of the structure in this field to point to `FSCatalogInfo` structures containing the values to match against.

`maximumItems`
On input, the maximum number of items to return for this call.

`actualItems`
On output, the actual number of items returned for this call.

`containerChanged`
On output, a Boolean value indicating whether the container's contents have changed. If `true`, the container's contents changed since the previous `PBCatalogSearchSync` call. Objects may still be returned even though the container changed. Note that if the container has changed, then the total set of items returned may be incorrect; some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`
On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

`catalogInfo`
On output, a pointer to an array of `FSCatalogInfo` (page 212) structures; one for each found item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here.

`refs`
On output, a pointer to an array of `FSRef` (page 223) structures; one for each returned item. On input, if you want an `FSRef` for each item found, pass a pointer to an array of `maximumItems FSRef` structures. Otherwise, pass `NULL`.

`names`
On output, a pointer to an array of filenames; one for each returned item. On input, if you want the Unicode filename for each item found, pass a pointer to an array of `maximumItems HFSUniStr255` (page 241) structures. Otherwise, pass `NULL`.

A single search may span more than one call to `PBCatalogSearchSync`. The call may complete with no error before scanning the entire volume. This typically happens because the time limit (`searchTime`) has been reached or `maximumItems` items have been returned. If the search is not completed, you can continue the search by making another call to `PBCatalogSearchSync` and passing the updated iterator returned by the previous call in the `iterator` field.

Before calling this function, you should determine that it is present, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCloseForkAsync

Closes an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBCloseForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam`.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

The `PBCloseForkAsync` function causes all data written to the fork to be written to disk, in the same manner as the `PBFlushForkAsync` (page 133) function, before it closes the fork.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCloseForkSync

Closes an open fork.

```
OSErr PBCloseForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam`.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant field of the parameter block is:

`forkRefNum`

On input, the reference number of the fork to close. After the call to this function, the reference number in this parameter is invalid.

The `PBCloseForkSync` function causes all data written to the fork to be written to disk, in the same manner as the `PBFlushForkSync` (page 133) function, before it closes the fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCloseIteratorAsync

Closes a catalog iterator.

```
void PBCloseIteratorAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`iterator`

On input, the catalog iterator to close. `PBCloseIteratorAsync` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCloseIteratorSync

Closes a catalog iterator.

```
OSErr PBCloseIteratorSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant field of the parameter block is:

`iterator`
On input, the catalog iterator to close. `PBCloseIteratorSync` releases memory and other system resources used by the iterator, making the iterator invalid. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCompareFSRefsAsync

Determines whether two `FSRef` structures refer to the same file or directory.

```
void PBCompareFSRefsAsync (
   FSRefParam *paramBlock
);
```

Parameters
paramBlock
A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information about completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. If the two `FSRef` structures refer to the same file or directory, then `noErr` is returned. If they refer to objects on different volumes, then `diffVolErr` is returned. If they refer to different files or directories on the same volume, then `errFSRefsDifferent` is returned. This call may return other errors, including `nsvErr`, `fnfErr`, `dirNFErr`, and `volOffLinErr`. See "File Manager Result Codes".

`ref`
On input, a pointer to the first `FSRef` to compare. See `FSRef` (page 223) for a description of the `FSRef` data type.

`parentRef`
On input, a pointer to the second `FSRef` to compare.

You must use `FSCompareFSRefs` (page 48) , or one of the corresponding parameter block functions, `PBCompareFSRefsSync` (page 121) and `PBCompareFSRefsAsync`, to compare `FSRef` structures. It is not possible to compare the `FSRef` structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Some volume formats may be able to tell that two FSRef structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return errFSRefsDifferent even if one or both objects no longer exist. Similarly, if the FSRef structures are for objects on different volumes, the File Manager will return diffVolErr even if one or both volumes are no longer mounted.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Files.h

PBCompareFSRefsSync

Determines whether two FSRef structures refer to the same file or directory.

```
OSErr PBCompareFSRefsSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 223) for a description of the FSRefParam data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the two FSRef structures refer to the same file or directory, then noErr is returned. If they refer to objects on different volumes, then diffVolErr is returned. If they refer to different files or directories on the same volume, then errFSRefsDifferent is returned. This function may return other errors, including nsvErr, fnfErr, dirNFErr, and volOffLinErr.

Discussion
The relevant fields of the parameter block are:

ref
On input, a pointer to the first FSRef to compare. See FSRef (page 223) for a description of the FSRef data type.

parentRef
On input, a pointer to the second FSRef to compare.

You must use FSCompareFSRefs (page 48) , or one of the corresponding parameter block functions, PBCompareFSRefsSync and PBCompareFSRefsAsync (page 120) , to compare FSRef structures. It is not possible to compare the FSRef structures directly since some bytes may be uninitialized, case-insensitive text, or contain hint information.

Some volume formats may be able to tell that two FSRef structures would refer to two different files or directories, without having to actually find those objects. In this case, the volume format may return errFSRefsDifferent even if one or both objects no longer exist. Similarly, if the FSRef structures are for objects on different volumes, the File Manager will return diffVolErr even if one or both volumes are no longer mounted.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBCreateDirectoryUnicodeAsync

Creates a new directory (folder) with a Unicode name.

```
void PBCreateDirectoryUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information about completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ref`
On input, a pointer to an `FSRef` (page 223) for the parent directory where the new directory is to be created.

`nameLength`
On input, the number of Unicode characters in the new directory's name.

`name`
On input, a pointer to the Unicode name of the new directory.

`whichInfo`
On input, a bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this field.

`catInfo`
On input, a pointer to the `FSCatalogInfo` (page 212) structure which specifies the values of the new directory's catalog information fields. Specify which fields to set in the `whichInfo` field. Specify `NULL` if you do not wish to set catalog information for the new directory.

`newRef`
On output, a pointer to the `FSRef` for the new directory. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`
On output, a pointer to the `FSSpec` (page 226) for the new directory. If you do not want the `FSSpec` returned, pass `NULL` on input.

`ioDirID`
On output, the directory ID of the new directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

You may optionally set catalog information for the new directory using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 101) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCreateDirectoryUnicodeSync

Creates a new directory (folder) with a Unicode name.

```
OSErr PBCreateDirectoryUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ref`
On input, a pointer to an `FSRef` (page 223) for the parent directory where the new directory is to be created.

`nameLength`
On input, the number of Unicode characters in the new directory's name.

`name`
On input, a pointer to the Unicode name of the new directory.

`whichInfo`
On input, a bitmap specifying which catalog information fields to set for the new directory. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new directory, specify the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this field.

`catInfo`
On input, a pointer to the `FSCatalogInfo` (page 212) structure which specifies the values of the new directory's catalog information fields. Specify which fields to set in the `whichInfo` field. Specify `NULL` if you do not wish to set catalog information for the new directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`newRef`
On output, a pointer to the `FSRef` for the new directory. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`
On output, a pointer to the `FSSpec` (page 226) for the new directory. If you do not want the `FSSpec` returned, pass `NULL` on input.

`ioDirID`
On output, the directory ID of the new directory.

You may optionally set catalog information for the new directory using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 101) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162) , after creating the directory.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCreateFileUnicodeAsync

Creates a new file with a Unicode name.

```
void PBCreateFileUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ref`
On input, a pointer to an `FSRef` (page 223) for the directory where the file is to be created.

`nameLength`
On input, the number of Unicode characters in the file's name.

`name`
On input, a pointer to the Unicode name of the new file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set for the new file. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new file, pass the constant `kFSCatInfoNone` here. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this field.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 212) structure which specifies the values of the new file's catalog information fields. Specify which fields to set in the `whichInfo` field. This field is optional; specify `NULL` if you do not wish to set catalog information for the new file.

`newRef`

On output, a pointer to the `FSRef` for the new file. If you do not want the `FSRef` returned, pass `NULL` on input.

`spec`

On output, a pointer to the `FSSpec` for the new file. If you do not want the `FSSpec` (page 226) returned, pass `NULL` on input.

You may optionally set catalog information for the file using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 101) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162) , after creating the file.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `PBCreateFileUnicodeAsync` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateAsync` (page 438). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCreateFileUnicodeSync

Creates a new file with a Unicode name.

```
OSErr PBCreateFileUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ref`
On input, a pointer to an `FSRef` (page 223) for the directory where the file is to be created.

`nameLength`
On input, the number of Unicode characters in the file's name.

`name`
On input, a pointer to the Unicode name of the new file.

`whichInfo`
On input, a bitmap specifying which catalog information fields to set for the new file. Specify the values for these fields in the `catInfo` field. If you do not wish to set catalog information for the new file, pass the constant `kFSCatInfoNone` here. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits defined for this field.

`catInfo`
On input, a pointer to the `FSCatalogInfo` (page 212) structure which specifies the values of the new file's catalog information fields. Specify which fields to set in the `whichInfo` field. This field is optional; specify `NULL` if you do not wish to set catalog information for the new file.

`newRef`
On output, a pointer to the `FSRef` for the new file. If you do not want the `FSRef` returned, set this field to `NULL` on input.

`spec`
On output, a pointer to the `FSSpec` (page 226) for the new file. If you do not want the `FSSpec` returned, set this field to `NULL` on input.

You may optionally set catalog information for the new file using the `whichInfo` and `catInfo` fields; this is equivalent to calling `FSSetCatalogInfo` (page 101) , or one of the corresponding parameter block functions, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162) , after creating the file.

If possible, you should set the `textEncodingHint` field of the catalog information structure specified in the `catInfo` field. This will be used by the volume format when converting the Unicode filename to other encodings.

Special Considerations

If the `PBCreateFileUnicodeSync` function is present, but is not implemented by a particular volume, the File Manager will emulate this function by making the appropriate call to `PBHCreateSync` (page 439). However, if the function is not directly supported by the volume, you will not be able to use the long Unicode filenames, or other features added with HFS Plus.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBCreateForkAsync

Creates a named fork for a file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBCreateForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkAsync` returns `errFSBadForkName` or `errFSNameTooLong`.

`ref`

On input, a pointer to an `FSRef` (page 223) specifying the file or directory.

`forkNameLength`

On input, the length of the Unicode name of the new fork.

`forkName`

On input, a pointer to the Unicode name of the fork.

A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBCreateForkSync

Creates a named fork for a file or directory.

```
OSErr PBCreateForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329). . If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkSync` returns `errFSBadForkName` or `errFSNameTooLong`.

Discussion
The relevant fields of the parameter block are:

`ioResult`
: On output, the result code of the function. If the named fork already exists, the function returns `errFSForkExists`. If the fork name is syntactically invalid or otherwise unsupported for the given volume, `PBCreateForkAsync` returns `errFSBadForkName` or `errFSNameTooLong`.

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory.

`forkNameLength`
: On input, the length of the Unicode name of the new fork.

`forkName`
: On input, a pointer to the Unicode name of the fork.

A newly created fork has zero length (that is, its logical end-of-file is zero). The data and resource forks of a file are automatically created and deleted as needed. This is done for compatibility with older APIs, and because data and resource forks are often handled specially. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), an attempt to create that fork when a zero-length fork already exists should return `noErr`; if a non-empty fork already exists then `errFSForkExists` should be returned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBDeleteForkAsync

Deletes a named fork of a file or directory.

```
void PBDeleteForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
On output, the result code of the function. If the named fork does not exist, the function returns `errFSForkNotFound`.

`ref`
On input, a pointer to an `FSRef` (page 223) for the file or directory from which to delete the fork.

`forkNameLength`
On input, the length of the fork's Unicode name.

`forkName`
On input, a pointer to the Unicode name of the fork to delete.

The `permissions`, `forkRefNum`, `positionMode`, and `positionOffset` fields of the parameter block may be modified by this call.

Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBDeleteForkSync

Deletes a named fork from a file or directory.

```
OSErr PBDeleteForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the named fork does not exist, the function returns `errFSForkNotFound`.

Discussion
The relevant fields of the parameter block are:

`ref`
On input, a pointer to an `FSRef` (page 223) for the file or directory from which to delete the fork.

`forkNameLength`
On input, the length of the fork's Unicode name.

`forkName`
On input, a pointer to the Unicode name of the fork to delete.

The `permissions`, `forkRefNum`, `positionMode`, and `positionOffset` fields of the parameter block may be modified by this call.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Any storage allocated to the fork is released. If a given fork always exists for a given volume format (such as data and resource forks for HFS and HFS Plus, or data forks for most other volume formats), this is equivalent to setting the logical size of the fork to zero.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBDeleteObjectAsync

Deletes a file or an empty directory.

```
void PBDeleteObjectAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

A result code. See "File Manager Result Codes" (page 329). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

`ref`

On input, a pointer to the `FSRef` (page 223) for the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBDeleteObjectSync

Deletes a file or an empty directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBDeleteObjectSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you attempt to delete a folder for which there is an open catalog iterator, this function succeeds and returns `noErr`. Iteration, however, will continue to work until the iterator is closed.

Discussion

The relevant field of the parameter block is:

`ref`

On input, a pointer to the `FSRef` (page 223) for the file or directory to be deleted. If the object to be deleted is a directory, it must be empty (it must contain no files or folders).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBExchangeObjectsAsync

Swaps the contents of two files.

```
void PBExchangeObjectsAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ref`

On input, a pointer to an `FSRef` (page 223) for the first file.

`parentRef`

On input, a pointer to an `FSRef` for the second file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

The PBExchangeObjectsAsync function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, FSSpec, or FSRef that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

Availability
Available in Mac OS X v10.0 and later.

Declared In
Files.h

PBExchangeObjectsSync

Swaps the contents of two files.

```
OSErr PBExchangeObjectsSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See FSRefParam (page 223) for a description of the FSRefParam data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ref

On input, a pointer to an FSRef (page 223) for the first file.

parentRef

On input, a pointer to an FSRef for the second file.

The PBExchangeObjectsSync function allows programs to implement a "safe save" operation by creating and writing a complete new file and swapping the contents. An alias, FSSpec, or FSRef that refers to the old file will now access the new data. The corresponding information in in-memory data structures are also exchanged.

Either or both files may have open access paths. After the exchange, the access path will refer to the opposite file's data (that is, to the same data it originally referred, which is now part of the other file).

Availability
Available in Mac OS X v10.0 and later.

Declared In
Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBFlushForkAsync

Causes all data written to an open fork to be written to disk.

```
void PBFlushForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for more information on the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the fork to flush.

The `PBFlushForkAsync` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `PBFlushForkAsync` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `PBFlushForkAsync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBFlushForkSync

Causes all data written to an open fork to be written to disk.

```
OSErr PBFlushForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for more information on the `FSForkIOParam` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant field of the parameter block is:

`forkRefNum`
On input, the reference number of the fork to flush.

The `PBFlushForkSync` function causes the actual fork contents to be written to disk, as well as any other volume structures needed to access the fork. On HFS and HFS Plus, this includes the catalog, extents, and attribute B-trees; the volume bitmap; and the volume header and alternate volume header (the MDB and alternate MDB on HFS volumes), as needed.

On volumes that do not support `PBFlushForkSync` directly, the entire volume is flushed to be sure all volume structures associated with the fork are written to disk.

You do not, need to use `PBFlushForkSync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBFlushVolumeAsync

For the specified volume, writes all open and modified files in the current process to permanent storage.

```
OSStatus PBFlushVolumeAsync (
    FSRefParamPtr paramBlock
);
```

Parameters
`paramBlock`
A parameter block containing the volume reference number of the volume to flush. See `FSRefParam` (page 223).

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`Files.h`

PBFlushVolumeSync

For the specified volume, writes all open and modified files in the current process to permanent storage.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSStatus PBFlushVolumeSync (
    FSRefParamPtr paramBlock
);
```

Parameters

paramBlock

A parameter block containing the volume reference number of the volume to flush. See `FSRefParam` (page 223).

Return Value

A result code. See "File Manager Result Codes" (page 329).

Availability

Available in Mac OS X v10.5 and later.

Declared In

`Files.h`

PBFSCopyFileAsync

Duplicates a file and optionally renames it.

```
OSStatus PBFSCopyFileAsync (
    FSRefParamPtr paramBlock
);
```

Availability

Available in Mac OS X v10.5 and later.

Declared In

`Files.h`

PBFSCopyFileSync

Duplicates a file and optionally renames it.

```
OSStatus PBFSCopyFileSync (
    FSRefParamPtr paramBlock
);
```

Availability

Available in Mac OS X v10.5 and later.

Declared In

`Files.h`

PBGetCatalogInfoAsync

Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBGetCatalogInfoAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See FSRefParam (page 223) fro s description of the FSRefParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ref

On input, a pointer to an FSRef (page 223) specifying the file or directory for which to retrieve information.

whichInfo

On input, a bitmap specifying the catalog information fields to return. If you don't want any catalog information, set whichInfo to the constant kFSCatInfoNone. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

catInfo

On output, a pointer to an FSCatalogInfo (page 212) structure containing the information about the file or directory. Only the information specified in the whichInfo field is returned. If you don't want any catalog information, pass NULL here.

spec

On output, a pointer to the FSSpec (page 226) for the file or directory. This output is optional; if you do not wish the FSSpec returned, pass NULL here.

parentRef

On output, a pointer to the FSRef for the object's parent directory. This output is optional; if you do not wish the parent directory returned, pass NULL here. If the object specified in the ref field is a volume's root directory, then the FSRef returned in this field will not be a valid FSRef, since the root directory has no parent object.

outName

On output, a pointer to the Unicode name of the file or directory. On input, pass a pointer to an HFSUniStr255 (page 241) structure if you wish the name returned; otherwise, pass NULL.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBGetCatalogInfoBulkAsync

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBGetCatalogInfoBulkAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. When all of the iterator's objects have been returned, the call will return `errFSNoMoreItems`.

`iterator`

On input, the iterator to use. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, the iterator must be created with the `kFSIterateFlat` option. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

`maximumItems`

On input, the maximum number of items to return for this call.

`actualItems`

On output, the actual number of items found for this call.

`containerChanged`

On output, a value indicating whether or not the container's contents have changed since the previous `PBGetCatalogInfoBulkAsync` call. If `true`, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`

On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. For a description of the bits in this field, see "Catalog Information Bitmap Constants" (page 277).

`catalogInfo`

On output, a pointer to an array of catalog information structures; one for each returned item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here. See `FSCatalogInfo` (page 212) for a description of the `FSCatalogInfo` data type.

`refs`

On input, a pointer to an array of `maximumItems FSRef` (page 223) structures. On output, an `FSRef` is filled out for each returned item. This field is optional; if you do not wish any `FSRef` structures returned, pass `NULL` here.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`names`

On output, a pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this field to point to an array of `maximumItems HFSUniStr255` (page 241) structures. Otherwise, set it to `NULL`.

`specs`

On input, a pointer to an array of `maximumItems FSSpec` structures. On output, an `FSSpec` structure is filled out for each returned item. This field is optional; if you do not wish any `FSSpec` structures returned, pass `NULL` here.

The `PBGetCatalogInfoBulkAsync` call may complete and return `noErr` with fewer than `maximumItems` items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make `PBGetCatalogInfoBulkSync` calls using the same iterator.

Before calling this function, you should determine whether it is available, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBGetCatalogInfoBulkSync

Returns information about one or more objects from a catalog iterator. This function can return information about multiple objects in a single call.

```
OSErr PBGetCatalogInfoBulkSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). When all of the iterator's objects have been returned, the call will return `errFSNoMoreItems`.

Discussion
The relevant fields of the parameter block are:

`iterator`

On input, the iterator to use. You can obtain a catalog iterator with the function `FSOpenIterator` (page 88) , or with one of the related parameter block calls, `PBOpenIteratorSync` (page 157) and `PBOpenIteratorAsync` (page 156). Currently, the iterator must be created with the `kFSIterateFlat` option. See `FSIterator` (page 221) for a description of the `FSIterator` data type.

`maximumItems`

On input, the maximum number of items to return for this call.

`actualItems`

On output, the actual number of items found for this call.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`containerChanged`

On output, a value indicating whether or not the container's contents have changed since the previous `PBGetCatalogInfoBulkSync` call. If `true`, the contents have changed. Objects may still be returned, even though the container has changed. If so, note that if the container has changed, then the total set of items returned may be incorrect: some items may be returned multiple times, and some items may not be returned at all.

`whichInfo`

On input, a bitmap specifying the catalog information fields to return for each item. If you don't wish any catalog information returned, pass the constant `kFSCatInfoNone` in this field. For a description of the bits in this field, see "Catalog Information Bitmap Constants" (page 277).

`catalogInfo`

On output, a pointer to an array of catalog information structures; one for each returned item. On input, the `catalogInfo` field should point to an array of `maximumItems` catalog information structures. This field is optional; if you do not wish any catalog information returned, pass `NULL` here. See `FSCatalogInfo` (page 212) for a description of the `FSCatalogInfo` data type.

`refs`

On input, a pointer to an array of `maximumItems HFSUniStr255` (page 241) structures. On output, an `FSRef` is filled out for each returned item. This field is optional; if you do not wish any `FSRef` structures returned, pass `NULL` here.

`names`

On output, a pointer to an array of names; one for each returned item. If you want the Unicode name for each item found, set this field to point to an array of `maximumItems HFSUniStr255` (page 241) structures. Otherwise, set it to `NULL`.

`specs`

On input, a pointer to an array of `maximumItems FSSpec` structures. On output, an `FSSpec` structure is filled out for each returned item. This field is optional; if you do not wish any `FSSpec` structures returned, pass `NULL` here.

The `PBGetCatalogInfoBulkSync` call may complete and return `noErr` with fewer than `maximumItems` items returned. This may be due to various reasons related to the internal implementation. In this case, you may continue to make `PBGetCatalogInfoBulkSync` calls using the same iterator.

Before calling this function, you should determine whether it is available, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBGetCatalogInfoSync

Returns catalog information about a file or directory. You can use this function to map from an `FSRef` to an `FSSpec`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBGetCatalogInfoSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 223) specifying the file or directory for which to retrieve information.

`whichInfo`

On input, a bitmap specifying the catalog information fields to return. If you don't want any catalog information, set `whichInfo` to the constant `kFSCatInfoNone`. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

`catInfo`

On output, a pointer to an `FSCatalogInfo` (page 212) structure containing the information about the file or directory. Only the information specified in the `whichInfo` field is returned. If you don't want any catalog information, pass `NULL` here.

`spec`

On output, a pointer to the `FSSpec` (page 226) for the file or directory. This output is optional; if you do not wish the `FSSpec` returned, pass `NULL` here.

`parentRef`

On output, a pointer to the `FSRef` for the object's parent directory. This output is optional; if you do not wish the parent directory returned, pass `NULL` here. If the object specified in the `ref` field is a volume's root directory, then the `FSRef` returned in this field will not be a valid `FSRef`, since the root directory has no parent object.

`outName`

On output, a pointer to the Unicode name of the file or directory. On input, pass a pointer to an `HFSUniStr255` (page 241) structure if you wish the name returned; otherwise, pass `NULL`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkCBInfoAsync

Returns information about a specified open fork, or about all open forks.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBGetForkCBInfoAsync (
   FSForkCBInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork control block parameter block. See FSForkCBInfoParam (page 216) for a description of the FSForkCBInfoParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

desiredRefNum

On input, if you want information on a specific fork, set this field to that fork's reference number. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number the field is unchanged on output. Pass zero in this field to iterate over all open forks; on output, this field contains the fork's reference number. You can limit this iteration to a specific volume with the volumeRefNum field.

volumeRefNum

On input, the volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this field. To iterate over all open forks on all volumes, set this field to the constant kFSInvalidVolumeRefNum. This field is ignored if you specify a fork reference number in the desiredRefNum parameter. Set desiredRefNum to zero if you wish to iterate over multiple forks. See FSVolumeRefNum (page 233) for a description of the FSVolumeRefNum data type.

iterator

On input, an iterator. If the desiredRefNum parameter is 0, the iterator maintains state between calls to PBGetForkCBInfoAsync. Set the iterator field to 0 before you begin iterating, on the first call to PBGetForkCBInfoAsync. On return, the iterator will be updated; pass this updated iterator in the iterator field of the next call to PBGetForkCBInfoAsync to continue iterating.

actualRefNum

On output, the actual reference number of the open fork that was found.

ref

On output, a pointer to the FSRef (page 223) for the file or directory that contains the fork. This information is optional; if you do not wish to the FSRef, set ref to NULL.

forkInfo

On output, a pointer to an FSForkInfo (page 218) structure containing information about the open fork. This information is optional; if you do not wish it returned, set forkInfo to NULL.

forkName

On output, a pointer to the name of the fork. This field is optional; if you do not wish the name returned, set forkName to NULL. See HFSUniStr255 (page 241) for a description of the HFSUniStr255 data type.

Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use FSGetForkCBInfo (page 71) , or one of the related parameter block functions, PBGetForkCBInfoSync (page 142) and PBGetForkCBInfoAsync, to access information about a fork control block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

Returning the fork information in the `forkInfo` field generally does not require a disk access; returning the information in the `ref` or `forkName` fields may cause disk access for some volume formats.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkCBInfoSync

Returns information about a specified open fork, or about all open forks.

```
OSErr PBGetForkCBInfoSync (
   FSForkCBInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork control block parameter block. See `FSForkCBInfoParam` (page 216) for a description of the `FSForkCBInfoParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you are iterating over multiple forks, the function returns `errFSNoMoreItems` if there are no more open forks to return.

Discussion

The relevant fields of the parameter block are:

`desiredRefNum`

On input, if you want information on a specific fork, set this field to that fork's reference number. If you pass a non-zero value in this parameter, the function attempts to get information on the fork specified by that reference number the field is unchanged on output. Pass zero in this field to iterate over all open forks; on output, this field contains the fork's reference number. You can limit this iteration to a specific volume with the `volumeRefNum` field.

`volumeRefNum`

On input, the volume to search, when iterating over multiple forks. To iterate over all open forks on a single volume, specify the volume reference number in this field. To iterate over all open forks on all volumes, set this field to the constant `kFSInvalidVolumeRefNum`. This field is ignored if you specify a fork reference number in the `desiredRefNum` parameter. Set `desiredRefNum` to zero if you wish to iterate over multiple forks. See `FSVolumeRefNum` (page 233) for a description of the `FSVolumeRefNum` data type.

`iterator`

On input, an iterator. If the `desiredRefNum` parameter is 0, the iterator maintains state between calls to `PBGetForkCBInfoSync`. Set the `iterator` field to 0 before you begin iterating, on the first call to `PBGetForkCBInfoSync`. On return, the iterator will be updated; pass this updated iterator in the `iterator` field of the next call to `PBGetForkCBInfoSync` to continue iterating.

`actualRefNum`

On output, the actual reference number of the open fork that was found.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ref`

On output, a pointer to the `FSRef` (page 223) for the file or directory that contains the fork. This information is optional; if you do not wish to the `FSRef`, set `ref` to `NULL`.

`forkInfo`

On output, a pointer to an `FSForkInfo` (page 218) structure containing information about the open fork. This information is optional; if you do not wish it returned, set `forkInfo` to `NULL`.

`forkName`

On output, a pointer to the name of the fork. This field is optional; if you do not wish the name returned, set `forkName` to `NULL`. See `HFSUniStr255` (page 241) for a description of the `HFSUniStr255` data type.

Carbon applications are no longer guaranteed access to the FCB table. Instead, applications should use `FSGetForkCBInfo` (page 71) , or one of the related parameter block functions, `PBGetForkCBInfoSync` and `PBGetForkCBInfoAsync` (page 140) , to access information about a fork control block.

Special Considerations

Returning the fork information in the `forkInfo` field generally does not require a disk access; returning the information in the `ref` or `forkName` fields may cause disk access for some volume formats.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBGetForkPositionAsync

Returns the current position of an open fork.

```
void PBGetForkPositionAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of a fork previously opened by the `FSOpenFork` (page 87) , `PBOpenForkSync` (page 155) , or `PBOpenForkAsync` (page 154) function.

`positionOffset`

On output, the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

Before calling the `PBGetForkPositionAsync` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `PBGetForkPositionAsync` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosAsync` (page 435); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkPositionSync

Returns the current position of an open fork.

```
OSErr PBGetForkPositionSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of a fork previously opened by the `FSOpenFork` (page 87) , `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154) function.

`positionOffset`

On output, the current position of the fork. The returned fork position is relative to the start of the fork (that is, it is an absolute offset in bytes).

Special Considerations

Before calling the `PBGetForkPositionSync` function, call the `Gestalt` function with the `gestaltFSAttr` selector to determine if `PBGetForkPositionSync` is available. If the function is available, but is not directly supported by a volume, the File Manager will automatically call `PBGetFPosSync` (page 436); however, you will not be able to determine the fork position of a named fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBGetForkSizeAsync

Returns the size of an open fork.

```
void PBGetForkSizeAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`forkRefNum`

On input, the reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 87) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionOffset`

On output, the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Special Considerations

To determine whether the `PBGetForkSizeAsync` function is present, call the `Gestalt` function. If `PBGetForkSizeAsync` is present, but is not directly supported by a volume, the File Manager will call `PBGetEOFAsync` (page 429); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetForkSizeSync

Returns the size of an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBGetForkSizeSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of the open fork. You can obtain this fork reference number with the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionOffset`

On output, the logical size (the logical end-of-file) of the fork, in bytes. The size returned is the total number of bytes that can be read from the fork; the amount of space actually allocated on the volume (the physical size) will probably be larger.

Special Considerations

To determine whether the `PBGetForkSizeSync` function is present, call the `Gestalt` function. If `PBGetForkSizeSync` is present, but is not directly supported by a volume, the File Manager will call `PBGetEOFSync` (page 430); however, you will not be able to determine the size of a fork other than the data or resource fork, or of a fork larger than 2 GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBGetVolumeInfoAsync

Returns information about a volume.

```
void PBGetVolumeInfoAsync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 231) for a description of the `FSVolumeInfoParam` data type.

Discussion

The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioVRefNum`
: On input, the volume whose information is to be returned. For information on a particular volume, pass that volume's reference number and set the `volumeIndex` field to 0. To index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum`.On output, the volume reference number of the volume. This is useful when indexing over all mounted volumes, when you have not specified a particular volume reference number on input.

`volumeIndex`
: On input, the index of the desired volume, or 0 to use the volume reference number in the `ioVRefNum` field.

`whichInfo`
: On input, a bitmap specifying which volume information fields to return in the `volumeInfo` field. If you don't want the information about the volume returned in the `volumeInfo` field, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this field.

`volumeInfo`
: On output, a pointer to the volume information, as described by the `FSVolumeInfo` (page 228) data type. If you don't want this output, set this field to `NULL`.

`volumeName`
: On output, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. Otherwise, pass a pointer to an `HFSUniStr255` (page 241) structure.

`ref`
: On output, a pointer to the `FSRef` (page 223) for the volume's root directory. If you do not wish the root directory returned, pass `NULL`.

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `ioVRefNum` field and set the `volumeIndex` field to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the `ioVRefNum` field and set `volumeIndex` to the index, starting at 1 with the first call to `PBGetVolumeInfoAsync`.

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 67) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 139) and `PBGetCatalogInfoAsync` (page 135).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBGetVolumeInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability

Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBGetVolumeInfoSync

Returns information about a volume.

```
OSErr PBGetVolumeInfoSync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 231) for a description of the `FSVolumeInfoParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioVRefNum`
On input, the volume whose information is to be returned. For information on a particular volume, pass that volume's reference number and set the `volumeIndex` field to 0. To index through the list of mounted volumes, pass the constant `kFSInvalidVolumeRefNum`.On output, the volume reference number of the volume. This is useful when indexing over all mounted volumes, when you have not specified a particular volume reference number on input.

`volumeIndex`
On input, the index of the desired volume, or 0 to use the volume reference number in the `ioVRefNum` field.

`whichInfo`
On input, a bitmap specifying which volume information fields to return in the `volumeInfo` field. If you don't want the information about the volume returned in the `volumeInfo` field, set `whichInfo` to `kFSVolInfoNone`. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this field.

`volumeInfo`
On output, a pointer to the volume information, as described by the `FSVolumeInfo` (page 228) data type. If you don't want this output, set this field to `NULL`.

`volumeName`
On output, a pointer to the Unicode name of the volume. If you do not wish the name returned, pass `NULL`. Otherwise, pass a pointer to an `HFSUniStr255` (page 241) structure.

`ref`
On output, a pointer to the `FSRef` (page 223) for the volume's root directory. If you do not wish the root directory returned, pass `NULL`.

You can specify a particular volume or index through the list of mounted volumes. To get information on a particular volume, pass the volume reference number of the desired volume in the `ioVRefNum` field and set the `volumeIndex` field to zero. To index through the list of mounted volumes, pass `kFSInvalidVolumeRefNum` in the `ioVRefNum` field and set `volumeIndex` to the index, starting at 1 with the first call to `PBGetVolumeInfoSync`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

To get information about the root directory of a volume, use the `FSGetCatalogInfo` (page 67) function, or one of the corresponding parameter block calls, `PBGetCatalogInfoSync` (page 139) and `PBGetCatalogInfoAsync` (page 135).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBGetVolumeInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBIterateForksAsync

Determines the name and size of every named fork belonging to a file or directory.

```
void PBIterateForksAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for more information on the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ref`

On input, a pointer to an `FSRef` (page 223) specifying the file or directory to iterate.

`forkIterator`

A pointer to a structure which maintains state between calls to `PBIterateForksAsync`. Before the first call, set the `initialize` field of this structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See `CatPositionRec` (page 187) for a description of the structure pointed to in this field.

`outForkName`

On output, a pointer to the Unicode name of the fork.

`positionOffset`

On output, the logical size of the fork, in bytes.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`allocationAmount`
On output, the fork's physical size (that is, the amount of space allocated on disk), in bytes.

Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBIterateForksSync

Determines the name and size of every named fork belonging to a file or directory.

```
OSErr PBIterateForksSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for more information on the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ref`
On input, a pointer to an `FSRef` (page 223) specifying the file or directory to iterate.

`forkIterator`
A pointer to a structure which maintains state between calls to `PBIterateForksSync`. Before the first call, set the `initialize` field of this structure to 0. The fork iterator will be updated after the call completes; the updated iterator should be passed into the next call. See `CatPositionRec` (page 187) for a description of the structure pointed to in this field.

`outForkName`
On output, a pointer to the Unicode name of the fork.

`positionOffset`
On output, the logical size of the fork, in bytes.

`allocationAmount`
On output, the fork's physical size (that is, the amount of space allocated on disk), in bytes.

Since information is returned about one fork at a time, several calls may be required to iterate through all the forks. There is no guarantee about the order in which forks are returned; the order may vary between iterations.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBMakeFSRefUnicodeAsync

Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

```
void PBMakeFSRefUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock
A pointer to a file system reference parameter block. See FSRefParam (page 223) for a description of the FSRefParam data type.

Discussion
The relevant fields of the parameter block are:

ioCompletion
On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult
On output, the result code of the function.

ref
On input, a pointer to the FSRef of the parent directory of the file or directory for which to create a new FSRef. See FSRef (page 223) for a description of the FSRef data type.

nameLength
On input, the length of the file or directory name.

name
On input, a pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to PBMakeFSRefUnicodeAsync.

textEncodingHint
On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant kTextEncodingUnknown, the File Manager will use a default value.

newRef
On output, if the function returns a result of noErr, a pointer to the new FSRef

Availability
Available in Mac OS X v10.0 and later.

Declared In
Files.h

PBMakeFSRefUnicodeSync

Constructs an FSRef for a file or directory, given a parent directory and a Unicode name.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBMakeFSRefUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to the `FSRef` of the parent directory of the file or directory for which to create a new `FSRef`. See `FSRef` (page 223) for a description of the `FSRef` data type.

`nameLength`

On input, the length of the file or directory name.

`name`

On input, a pointer to the Unicode name for the file or directory. The name must be a leaf name; partial or full pathnames are not allowed. If you have a partial or full pathname in Unicode, you will have to parse it yourself and make multiple calls to `PBMakeFSRefUnicodeSync`.

`textEncodingHint`

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

`newRef`

On output, if the function returns a result of `noErr`, a pointer to the new `FSRef`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBMoveObjectAsync

Moves a file or directory into a different directory.

```
void PBMoveObjectAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion

The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. If the `parentRef` field specifies a non-existent object, `dirNFErr` is returned; if it refers to a file, then `errFSNotAFolder` is returned. If the directory specified in `parentRef` is on a different volume than the file or directory indicated by the `ref` field, `diffVolErr` is returned.

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory to move.

parentRef
: On input, a pointer to an `FSRef` specifying the directory into which the file or directory given in the `ref` field will be moved.

`newRef`
: On output, a pointer to the new `FSRef` for the file or directory in its new location. This field is optional; if you do not wish the `FSRef` returned, pass `NULL` here.

Moving an object may change its `FSRef`. If you want to continue to refer to the object, you should pass a non- `NULL` pointer in the `newRef` field and use the `FSRef` returned there to refer to the object after the move. The original `FSRef` passed in the `ref` field may or may not be usable after the move. The `newRef` field may point to the same storage as the `parentRef` or `ref` fields.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBMoveObjectSync

Moves a file or directory into a different directory.

```
OSErr PBMoveObjectSync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If the the `parentRef` field of the parameter block specifies a non-existent object, `dirNFErr` is returned; if it refers to a file, `errFSNotAFolder` is returned. If the directory specified in the `parentRef` field is on a different volume than the file or directory indicated in the `ref` field, `diffVolErr` is returned.

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
: On output, the result code of the function. If the `parentRef` field specifies a non-existent object, `dirNFErr` is returned; if it refers to a file, then `errFSNotAFolder` is returned. If the directory specified in `parentRef` is on a different volume than the file or directory indicated by the `ref` field, `diffVolErr` is returned.

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory to move.

`parentRef`
: On input, a pointer to an `FSRef` specifying the directory into which the file or directory given in the `ref` field will be moved.

`newRef`
: On output, a pointer to the new `FSRef` for the file or directory in its new location. This field is optional; if you do not wish the `FSRef` returned, pass `NULL` here.

Moving an object may change its `FSRef`. If you want to continue to refer to the object, you should pass a non- `NULL` pointer in the `newRef` field and use the `FSRef` returned there to refer to the object after the move. The original `FSRef` passed in the `ref` field may or may not be usable after the move. The `newRef` field may point to the same storage as the `parentRef` or `ref` fields.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBOpenForkAsync

Opens any fork of a file or directory for streaming access.

```
void PBOpenForkAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. On some file systems, `PBOpenForkAsync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory that owns the fork to open.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`forkNameLength`
: On input, the length of the fork's Unicode name.

`forkName`
: On input, a pointer to the Unicode name of the fork to open. You can obtain the string constants for the data and resource fork names using the `FSGetDataForkName` (page 70) and `FSGetResourceForkName` (page 73) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

`permissions`
: On input, a constant indicating the type of access that you wish to have to the fork via the returned fork reference. This parameter is the same as the `permission` parameter passed to the `FSpOpenDF` and `FSpOpenRF` functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 295).

`forkRefNum`
: On output, the fork reference number for accessing the open fork.

If you wish to access named forks or forks larger than 2GB you must use the `FSOpenFork` function or one of the corresponding parameter block calls, `PBOpenForkSync` and `PBOpenForkAsync`. To determine if the `PBOpenForkSync` function is present, call the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBOpenForkSync

Opens any fork of a file or directory for streaming access.

```
OSErr PBOpenForkSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). On some file systems, `PBOpenForkSync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion
The relevant fields of the parameter block are:

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory that owns the fork to open.

`forkNameLength`
: On input, the length of the fork's Unicode name.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`forkName`
: On input, a pointer to the Unicode name of the fork to open. You can obtain the string constants for the data and resource fork names using the `FSGetDataForkName` (page 70) and `FSGetResourceForkName` (page 73) functions. All volume formats should support data and resource forks; other named forks may be supported by some volume formats.

`permissions`
: On input, a constant indicating the type of access that you wish to have to the fork via the returned fork reference. This parameter is the same as the `permission` parameter passed to the `FSpOpenDF` and `FSpOpenRF` functions. For a description of the types of access which you can request, see "File Access Permission Constants" (page 295).

`forkRefNum`
: On output, the fork reference number for accessing the open fork.

If you wish to access named forks or forks larger than 2GB you must use the `FSOpenFork` function or one of the corresponding parameter block calls, `PBOpenForkSync` and `PBOpenForkAsync`. To determine if the `PBOpenForkSync` function is present, call the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBOpenIteratorAsync

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

```
void PBOpenIteratorAsync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

paramBlock
: A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`iterator`
: On output, the new `FSIterator` (page 221). You can pass this iterator to the `FSGetCatalogInfoBulk` (page 68) or `FSCatalogSearch` (page 45) functions and their parameter block-based counterparts. The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`iteratorFlags`

On input, a set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 310) for a description of the flags defined for this field. Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the `kFSIterateSubtree` flag in this field returns `errFSBadIteratorFlags`. To determine whether subtree iterators are supported, check that the `bSupportsSubtreeIterators` bit returned by `PBHGetVolParmsAsync` (page 517) is set.

`container`

On input, a pointer to an `FSRef` (page 223) for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate).

Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the `FSCloseIterator` (page 48) function, or one of the related parameter block functions, `PBCloseIteratorSync` (page 119) and `PBCloseIteratorAsync` (page 119) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBOpenIteratorSync

Creates a catalog iterator that can be used to iterate over the contents of a directory or volume.

```
OSErr PBOpenIteratorSync (
   FSCatalogBulkParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a catalog information parameter block. See `FSCatalogBulkParam` (page 210) for a description of the `FSCatalogBulkParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`iterator`

On output, the new `FSIterator` (page 221). You can pass this iterator to the `FSGetCatalogInfoBulk` (page 68) or `FSCatalogSearch` (page 45) functions and their parameter block-based counterparts. The iterator is automatically initialized so that the next use of the iterator returns the first item. The order that items are returned in is volume format dependent and may be different for two different iterators created with the same container and flags.

`iteratorFlags`

On input, a set of flags which controls whether the iterator iterates over subtrees or just the immediate children of the container. See "Iterator Flags" (page 310) for a description of the flags defined for this

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

field. Iteration over subtrees which do not originate at the root directory of a volume are not currently supported, and passing the `kFSIterateSubtree` flag in this field returns `errFSBadIteratorFlags`. To determine whether subtree iterators are supported, check that the `bSupportsSubtreeIterators` bit returned by `PBHGetVolParmsSync` (page 518) is set.

`container`

On input, a pointer to an `FSRef` (page 223) for the directory to iterate. The set of items to iterate over can either be the objects directly contained in the directory, or all items directly or indirectly contained in the directory (in which case, the specified directory is the root of the subtree to iterate).

Catalog iterators must be closed when you are done using them, whether or not you have iterated over all the items. Iterators are automatically closed upon process termination, just like open files. However, you should use the `FSCloseIterator` (page 48) function, or one of the related parameter block functions, `PBCloseIteratorSync` (page 119) and `PBCloseIteratorAsync` (page 119) , to close an iterator to free up any system resources allocated to the iterator.

Before calling this function, you should check that it is present, by calling the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBReadForkAsync

Reads data from an open fork.

```
void PBReadForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

`forkRefNum`

On input, the reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 87) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`

On input, a constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

location. The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently. If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches. See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching the read.

`positionOffset`
On input, the offset from the base location for the start of the read.

`requestCount`
On input, the number of bytes to read. The value that you pass in this field should be greater than zero.

`buffer`
A pointer to the buffer where the data will be returned.

`actualCount`
On output, the number of bytes actually read. The value in this field should be equal to the value in the `requestCount` field unless there was an error during the read operation.

`PBReadForkAsync` reads data starting at the position specified by the `positionMode` and `positionOffset` fields. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` field and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBReadForkSync

Reads data from an open fork.

```
OSErr PBReadForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329). If there are fewer than `requestCount` bytes from the specified position to the logical end-of-file, then all of those bytes are read, and `eofErr` is returned.

Discussion
The relevant fields of the parameter block are:

`forkRefNum`
On input, the reference number of the fork to read from. You should have previously opened this fork using the `FSOpenFork` (page 87) call, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`
On input, a constant specifying the base location within the fork for the start of the read. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being read should or should not be cached. Caching reads appropriately can be important in ensuring that your program access files efficiently. If you add the `forceReadMask` constant to the value you pass in this parameter, this tells the File Manager to force the data to be read directly from the disk. This is different from adding the `noCacheMask` constant since `forceReadMask` tells the File Manager to flush the appropriate part of the cache first, then ignore any data already in the cache. However, data that is read may be placed in the cache for future reads. The `forceReadMask` constant is also passed to the device driver, indicating that the driver should avoid reading from any device caches. See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching the read.

`positionOffset`
On input, the offset from the base location for the start of the read.

`requestCount`
On input, the number of bytes to read. The value that you pass in this field should be greater than zero.

`buffer`
A pointer to the buffer where the data will be returned.

`actualCount`
On output, the number of bytes actually read. The value in this field should be equal to the value in the `requestCount` field unless there was an error during the read operation.

`PBReadForkSync` reads data starting at the position specified by the `positionMode` and `positionOffset` fields. The function reads up to `requestCount` bytes into the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte read (that is, the initial position plus `actualCount`).

To verify that data previously written has been correctly transferred to disk, read it back in using the `forceReadMask` constant in the `positionMode` field and compare it with the data you previously wrote.

When reading data from a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your read requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBRenameUnicodeAsync

Renames a file or folder.

```
void PBRenameUnicodeAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See FSRefParam (page 223) for a description of the FSRefParam data type.

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ref

On input, a pointer to an FSRef (page 223) for the file or directory to rename.

nameLength

On input, the length of the new name in Unicode characters.

name

On input, a pointer to the new Unicode name of the file or directory.

textEncodingHint

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant kTextEncodingUnknown, the File Manager will use a default value.

newRef

On output, a a pointer to the new FSRef for the file or directory. This field is optional; if you do not wish the FSRef returned, pass NULL.

Because renaming an object may change its FSRef, you should pass a non- NULL pointer in the newRef field and use the FSRef returned there to access the object after the renaming, if you wish to continue to refer to the object. The FSRef passed in the ref field may or may not be usable after the object is renamed. The FSRef returned in the newRef field may point to the same storage as the FSRef passed in ref.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Files.h

PBRenameUnicodeSync

Renames a file or folder.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBRenameUnicodeSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 223) for the file or directory to rename.

`nameLength`

On input, the length of the new name in Unicode characters.

`name`

On input, a pointer to the new Unicode name of the file or directory.

`textEncodingHint`

On input, the suggested text encoding to use when converting the Unicode name of the file or directory to some other encoding. If you pass the constant `kTextEncodingUnknown`, the File Manager will use a default value.

`newRef`

On output, a a pointer to the new `FSRef` for the file or directory. This field is optional; if you do not wish the `FSRef` returned, pass `NULL`.

Because renaming an object may change its `FSRef`, you should pass a non- `NULL` pointer in the `newRef` field and use the `FSRef` returned there to access the object after the renaming, if you wish to continue to refer to the object. The `FSRef` passed in the `ref` field may or may not be usable after the object is renamed. The `FSRef` returned in the `newRef` field may point to the same storage as the `FSRef` passed in `ref`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetCatalogInfoAsync

Sets the catalog information about a file or directory.

```
void PBSetCatalogInfoAsync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ref`
: On input, a pointer to an `FSRef` (page 223) specifying the file or directory whose information is to be changed.

`whichInfo`
: On input, a bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` field. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

  To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this field. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `PBSetCatalogInfoAsync` cannot set the user and group IDs, it returns an error.

`catInfo`
: On input, a pointer to the `FSCatalogInfo` (page 212) structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

  - `createDate`
  - `contentModDate`
  - `attributeModDate`
  - `accessDate`
  - `backupDate`
  - `permissions`
  - `finderInfo`
  - `extFinderInfo`
  - `textEncodingHint`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBSetCatalogInfoSync

Sets the catalog information about a file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBSetCatalogInfoSync (
   FSRefParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a file system reference parameter block. See `FSRefParam` (page 223) fro s description of the `FSRefParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ref`

On input, a pointer to an `FSRef` (page 223) specifying the file or directory whose information is to be changed.

`whichInfo`

On input, a bitmap specifying which catalog information fields to set. Only some of the catalog information fields may be set. These fields are given by the constant `kFSCatInfoSettableInfo`; no other bits may be set in the `whichInfo` field. See "Catalog Information Bitmap Constants" (page 277) for a description of the bits in this field.

To set the user ID (UID) and group ID (GID), specify the `kFSCatInfoSetOwnership` flag in this field. The File Manager attempts to set the user and group ID to the values specified in the `permissions` field of the catalog information structure. If `PBSetCatalogInfoSync` cannot set the user and group IDs, it returns an error.

`catInfo`

On input, a pointer to the `FSCatalogInfo` (page 212) structure containing the new catalog information. Only some of the catalog information fields may be set. The fields which may be set are:

- `createDate`
- `contentModDate`
- `attributeModDate`
- `accessDate`
- `backupDate`
- `permissions`
- `finderInfo`
- `extFinderInfo`
- `textEncodingHint`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBSetForkPositionAsync

Sets the current position of an open fork.

```
void PBSetForkPositionAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. This function returns the result code `posErr` if you attempt to set the current position of the fork to an offset before the start of the file.

`forkRefNum`

On input, the reference number of a fork previously opened by the `FSOpenFork` (page 87) , `PBOpenForkSync` (page 155) , or `PBOpenForkAsync` (page 154) function.

`positionMode`

On input, a constant specifying the base location within the fork for the new position. If this field is equal to `fsAtMark`, then the `positionOffset` field is ignored. See "Position Mode Constants" (page 314) for a description of the constants you can use to specify the base location.

`positionOffset`

On input, the offset of the new position from the base location specified in the `positionMode` field.

Special Considerations

To determine if the `PBSetForkPositionAsync` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `PBSetForkPositionAsync` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosAsync` (page 485) function. However, if the volume does not directly support the `PBSetForkPositionAsync` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetForkPositionSync

Sets the current position of an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBSetForkPositionSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). This function returns the result code `posErr` if you attempt to set the current position of the fork to an offset before the start of the file.

Discussion

The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of a fork previously opened by the `FSOpenFork` (page 87) , `PBOpenForkSync` (page 155) , or `PBOpenForkAsync` (page 154) function.

`positionMode`

On input, a constant specifying the base location within the fork for the new position. If this field is equal to `fsAtMark`, then the `positionOffset` field is ignored. See "Position Mode Constants" (page 314) for a description of the constants you can use to specify the base location.

`positionOffset`

On input, the offset of the new position from the base location specified in the `positionMode` field.

Special Considerations

To determine if the `PBSetForkPositionSync` function is present, call the `Gestalt` function with the `gestaltFSAttr` selector. If the `PBSetForkPositionSync` function is present, but the volume does not directly support it, the File Manager will automatically call the `PBSetFPosSync` (page 486) function. However, if the volume does not directly support the `PBSetForkPositionSync` function, you can only set the file position for the data and resource forks, and you cannot grow these files beyond 2GB.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetForkSizeAsync

Changes the size of an open fork.

```
void PBSetForkSizeAsync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Discussion

The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

`forkRefNum`
: On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 87) function, or with one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`
: On input, a constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 314) for more information about the constants you can use to specify the base location.

`positionOffset`
: On input, the offset of the new size from the base location specified in the `positionMode` field.

The `PBSetForkSizeAsync` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` fields. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size. That is, the current position will be equal to the logical end of file.

Special Considerations

You do not need to check that the volume supports the `PBSetForkSizeAsync` function. If a volume does not support the `PBSetForkSizeAsync` function, but the `PBSetForkSizeAsync` function is present, the File Manager automatically calls the `PBSetEOFAsync` (page 483) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `PBSetForkSizeAsync` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `PBSetForkSizeAsync` function is present, call the `Gestalt` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBSetForkSizeSync

Changes the size of an open fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBSetForkSizeSync (
   FSForkIOParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). If there is not enough space on the volume to extend the fork, then `dskFulErr` is returned and the fork's size is unchanged.

Discussion

The relevant fields of the parameter block are:

`forkRefNum`

On input, the reference number of the open fork. You can obtain a fork reference number with the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`

On input, a constant indicating the base location within the fork for the new size. See "Position Mode Constants" (page 314) for more information about the constants you can use to specify the base location.

`positionOffset`

On input, the offset of the new size from the base location specified in the `positionMode` field.

The `PBSetForkSizeSync` function sets the logical end-of-file to the position indicated by the `positionMode` and `positionOffset` fields. The fork's new size may be less than, equal to, or greater than the fork's current size. If the fork's new size is greater than the fork's current size, then the additional bytes, between the old and new size, will have an undetermined value.

If the fork's current position is larger than the fork's new size, then the current position will be set to the new fork size. That is, the current position will be equal to the logical end-of-file.

Special Considerations

You do not need to check that the volume supports the `PBSetForkSizeSync` function. If a volume does not support the `PBSetForkSizeSync` function, but the `PBSetForkSizeSync` function is present, the File Manager automatically calls the `PBSetEOFSync` (page 484) function and translates between the calls appropriately.

Note, however, that if the volume does not support the `PBSetForkSizeSync` function, you can only access the data and resource forks, and you cannot grow the fork beyond 2GB. To check that the `PBSetForkSizeSync` function is present, call the `Gestalt` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetVolumeInfoAsync

Sets information about a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
void PBSetVolumeInfoAsync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 231) for a description of the `FSVolumeInfoParam` data type.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioVRefNum`

On input, the volume reference number of the volume whose information is to be changed.

`whichInfo`

On input, a bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this parameter.

`volumeInfo`

On input, the new volume information. See `FSVolumeInfo` (page 228) for more information about the volume information structure.

To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 101) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

PBSetVolumeInfoSync

Sets information about a volume.

```
OSErr PBSetVolumeInfoSync (
   FSVolumeInfoParam *paramBlock
);
```

Parameters

paramBlock

A pointer to a volume information parameter block. See `FSVolumeInfoParam` (page 231) for a description of the `FSVolumeInfoParam` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioVRefNum`
On input, the volume reference number of the volume whose information is to be changed.

`whichInfo`
On input, a bitmap specifying which information to set. Only some of the volume information fields may be set. The settable fields are given by the constant `kFSVolInfoSettableInfo`; no other bits may be set in `whichInfo`. The fields which may be set are the backupDate, finderInfo, and `flags` fields. See "Volume Information Bitmap Constants" (page 324) for a description of the bits in this parameter.

`volumeInfo`
On input, the new volume information. See `FSVolumeInfo` (page 228) for more information about the volume information structure.

To set information about the root directory of a volume, use the `FSSetCatalogInfo` (page 101) function, or one of the corresponding parameter block calls, `PBSetCatalogInfoSync` (page 163) and `PBSetCatalogInfoAsync` (page 162).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBWriteForkAsync

Writes data to an open fork.

```
void PBWriteForkAsync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam`.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`forkRefNum`
: On input, the reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`
: On input, a constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching.

`positionOffset`
: On input, the offset from the base location for the start of the write.

`requestCount`
: On input, the number of bytes to write.

`buffer`
: A pointer to a buffer containing the data to write.

`actualCount`
: On output, the number of bytes actually written. The value in the `actualCount` field will be equal to the value in the `requestCount` field unless there was an error during the write operation.

`PBWriteForkAsync` writes data starting at the position specified by the `positionMode` and `positionOffset` fields. The function attempts to write `requestCount` bytes from the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBWriteForkSync

Writes data to an open fork.

```
OSErr PBWriteForkSync (
    FSForkIOParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a fork I/O parameter block. See `FSForkIOParam` (page 219) for a description of the `FSForkIOParam`.

Return Value
A result code. See "File Manager Result Codes" (page 329). If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioResult`
: On output, the result code of the function. If there is not enough space on the volume to write `requestCount` bytes, then `dskFulErr` is returned.

`forkRefNum`
: On input, the reference number of the fork to which to write. You should have previously opened the fork using the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154).

`positionMode`
: On input, a constant specifying the base location within the fork for the start of the write. See "Position Mode Constants" (page 314) for a description of the constants which you can use to specify the base location. The caller can also use this parameter to hint to the File Manager whether the data being written should or should not be cached. See "Cache Constants" (page 275) for further description of the constants that you can use to indicate your preference for caching.

`positionOffset`
: On input, the offset from the base location for the start of the write.

`requestCount`
: On input, the number of bytes to write.

`buffer`
: A pointer to a buffer containing the data to write.

`actualCount`
: On output, the number of bytes actually written. The value in the `actualCount` field will be equal to the value in the `requestCount` field unless there was an error during the write operation.

`PBWriteForkSync` writes data starting at the position specified by the `positionMode` and `positionOffset` fields. The function attempts to write `requestCount` bytes from the buffer pointed to by the `buffer` field and sets the fork's current position to the byte immediately after the last byte written (that is, the initial position plus `actualCount`).

When writing data to a fork, it is important to pay attention to that way that your program accesses the fork, because this can have a significant performance impact. For best results, you should use an I/O size of at least 4KB and block align your write requests. In Mac OS X, you should align your requests to 4KB boundaries.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

PBXLockRangeAsync

Locks a range of bytes of the specified fork.

```
OSStatus PBXLockRangeAsync (
    FSRangeLockParamPtr paramBlock
);
```

Availability
Available in Mac OS X v10.4 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBXLockRangeSync

Locks a range of bytes of the specified fork.

```
OSStatus PBXLockRangeSync (
    FSRangeLockParamPtr paramBlock
);
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

PBXUnlockRangeAsync

Unlocks a range of bytes of the specified fork.

```
OSStatus PBXUnlockRangeAsync (
    FSRangeLockParamPtr paramBlock
);
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

PBXUnlockRangeSync

Unlocks a range of bytes of the specified fork.

```
OSStatus PBXUnlockRangeSync (
    FSRangeLockParamPtr paramBlock
);
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Callbacks by Task

File Operation Callbacks

`FSFileOperationStatusProcPtr` (page 175)
Defines a status callback function for an asynchronous file operation on an `FSRef` object.

`FSPathFileOperationStatusProcPtr` (page 176)
Defines a status callback function for an asynchronous file operation on an object specified with a pathname.

Miscellaneous Callbacks

`FNSubscriptionProcPtr` (page 174)
Callback delivered for directory notifications.

`FSVolumeEjectProcPtr` (page 177)

`FSVolumeMountProcPtr` (page 178)

`FSVolumeUnmountProcPtr` (page 179)

`IOCompletionProcPtr` (page 179)
Defines a pointer to a completion function. Your completion function is executed by the File Manager after the completion of an asynchronous File Manager function call.

Callbacks

FNSubscriptionProcPtr

Callback delivered for directory notifications.

```
typedef void (*FNSubscriptionProcPtr) (
    FNMessage message,
    OptionBits flags,
    void * refcon,
    FNSubscriptionRef subscription
);
```

If you name your function `MyFNSubscriptionProc`, you would declare it like this:

```
void MyFNSubscriptionProc (
    FNMessage message,
    OptionBits flags,
    void * refcon,
    FNSubscriptionRef subscription
);
```


2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Parameters

message

An indication of what happened.

flags

Options regarding the delivery of the notification; typically `kNilOptions`.

refcon

A pointer to a user reference supplied with subscription.

subscription

A subscription corresponding to this notification.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

FSFileOperationStatusProcPtr

Defines a status callback function for an asynchronous file operation on an `FSRef` object.

```
typedef void (*FSFileOperationStatusProcPtr) (
    FSFileOperationRef fileOp,
    const FSRef *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

If you name your function `MyFSFileOperationStatusProc`, you would declare it like this:

```
void MyFSFileOperationStatusProc (
    FSFileOperationRef fileOp,
    const FSRef *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

Parameters

fileOp

The file operation.

currentItem

A pointer to an `FSRef` variable. On output, the variable contains the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

stage

The current stage of the operation.

error

The current error status of the operation.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

statusDictionary

A dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 305). You are not responsible for releasing the dictionary.

info

A pointer to user-defined data associated with this operation.

Discussion

When you call `FSCopyObjectAsync` (page 50), `FSMoveObjectAsync` (page 83), or `FSMoveObjectToTrashAsync` (page 85), you can specify a status callback function of this type. The function you provide is called by the File Manager whenever the file operation changes stages (including failing due to an error), or as updated information is available limited by the status change interval of the operation. If you need to save any of the status information beyond the scope of the callback, you should make a copy of the information.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSPathFileOperationStatusProcPtr

Defines a status callback function for an asynchronous file operation on an object specified with a pathname.

```
typedef void (*FSPathFileOperationStatusProcPtr) (
    FSFileOperationRef fileOp,
    const char *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

If you name your function `MyFSPathFileOperationStatusProc`, you would declare it like this:

```
void MyFSPathFileOperationStatusProc (
    FSFileOperationRef fileOp,
    const char *currentItem,
    FSFileOperationStage stage,
    OSStatus error,
    CFDictionaryRef statusDictionary,
    void *info
);
```

Parameters

fileOp

The file operation.

currentItem

The UTF-8 pathname of the object currently being moved or copied. If the operation is complete, this parameter refers to the target (the new object corresponding to the source object in the destination directory).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

stage

The current stage of the operation.

error

The current error status of the operation.

statusDictionary

A dictionary with more detailed status information. For information about the contents of the dictionary, see "File Operation Status Dictionary Keys" (page 305). You are not responsible for releasing the dictionary.

info

A pointer to user-defined data associated with this operation.

Discussion

When you call `FSPathCopyObjectAsync` (page 90), `FSPathMoveObjectAsync` (page 94), or `FSPathMoveObjectToTrashAsync` (page 96), you can specify a status callback function of this type. The function you provide is called by the File Manager whenever the file operation changes stages (including failing due to an error), or as updated information is available limited by the status change interval of the operation. If you need to save any of the status information beyond the scope of the callback, you should make a copy of the information.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`Files.h`

FSVolumeEjectProcPtr

```
typedef void (*FSVolumeEjectProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

If you name your function `MyFSVolumeEjectProc`, you would declare it like this:

```
void MyFSVolumeEjectProc (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```


2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Parameters

volumeOp

clientData

err

volumeRefNum

dissenter

Availability
Available in Mac OS X v10.2 and later.

Declared In
Files.h

FSVolumeMountProcPtr

```
typedef void (*FSVolumeMountProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum mountedVolumeRefNum
);
```

If you name your function `MyFSVolumeMountProc`, you would declare it like this:

```
void MyFSVolumeMountProc (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum mountedVolumeRefNum
);
```

Parameters

volumeOp

clientData

err

mountedVolumeRefNum

Availability
Available in Mac OS X v10.2 and later.

Declared In
Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSVolumeUnmountProcPtr

```
typedef void (*FSVolumeUnmountProcPtr) (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

If you name your function `MyFSVolumeUnmountProc`, you would declare it like this:

```
void MyFSVolumeUnmountProc (
    FSVolumeOperation volumeOp,
    void * clientData,
    OSStatus err,
    FSVolumeRefNum volumeRefNum,
    pid_t dissenter
);
```

Parameters

volumeOp

clientData

err

volumeRefNum

dissenter

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Files.h`

IOCompletionProcPtr

Defines a pointer to a completion function. Your completion function is executed by the File Manager after the completion of an asynchronous File Manager function call.

```
typedef void (*IOCompletionProcPtr) (
    ParmBlkPtr paramBlock
);
```

If you name your function `MyIOCompletionProc`, you would declare it like this:

```
void MyIOCompletionProc (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the parameter block that was passed to the asynchronous File Manager function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value

Discussion
When you execute an asynchronous File Manager function (an `Async` function), you can specify a completion routine by passing the routine's address in the `ioCompletion` field of the parameter block passed to the function. Because you requested asynchronous execution, the File Manager places an I/O request in the file I/O queue and returns control to your application—possibly even before the actual I/O operation is completed. The File Manager takes requests from the queue one at a time and processes them meanwhile, your application is free to do other processing.

A function executed asynchronously returns control to your application with the result code `noErr` as soon as the call is placed in the file I/O queue. This result code does not indicate that the call has successfully completed, but simply indicates that the call was successfully placed in the queue. To determine when the call is actually completed, you can inspect the `ioResult` field of the parameter block. This field is set to a positive number when the call is made and set to the actual result code when the call is completed. If you specify a completion routine, it is executed after the result code is placed in `ioResult`.

The File Manager, when the File Sharing or AppleShare file server is active, will execute requests in arbitrary order. That means that if there is a request that depends on the completion of a previous request, it is an error for your program to issue the second request until the completion of the first request. For example, issuing a write request and then issuing a read request for the same data isn't guaranteed to read back what was written unless the read request isn't made until after the write request completes.

Request order can also change if a call results in a disk switch dialog to bring an offline volume back online.

Special Considerations

Because a completion routine is executed at interrupt time, it should not allocate, move, or purge memory (either directly or indirectly) and should not depend on the validity of handles to unlocked blocks.

If your completion routine uses application global variables, it must also ensure that register A5 contains the address of the boundary between your application global variables and your application parameters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

Data Types

AccessParam

Defines a parameter block used by low-level HFS file and directory access rights manipulation functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct AccessParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler3;
    short ioDenyModes;
    short filler4;
    SInt8 filler5;
    SInt8 ioACUser;
    long filler6;
    long ioACOwnerID;
    long ioACGroupID;
    long ioACAccess;
    long ioDirID;
};
typedef struct AccessParam AccessParam;
typedef AccessParam * AccessParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler3`
: Reserved.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDenyModes`
Access mode information.

`filler4`
Reserved.

`filler5`
Reserved.

`ioACUser`
The user's access rights for the specified directory.

`filler6`
Reserved.

`ioACOwnerID`
The owner ID.

`ioACGroupID`
The group ID.

`ioACAccess`
The directory access privileges.

`ioDirID`

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

AFPAlternateAddress

Defines a block of tagged addresses for AppleShare clients.

```
struct AFPAlternateAddress {
    UInt8 fVersion;
    UInt8 fAddressCount;
    UInt8 fAddressList[1];
};
typedef struct AFPAlternateAddress AFPAlternateAddress;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

AFPTagData

Defines a structure which contains tagged address information for AppleShare clients.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct AFPTagData {
    UInt8 fLength;
    UInt8 fType;
    UInt8 fData[1];
};
typedef struct AFPTagData AFPTagData;
```

Fields

`fLength`

The length, in bytes, of this data tag, including the `fLength` field itself. See "AFP Tag Length Constants" (page 271).

`fType`

The type of the data tag. See "AFP Tag Type Constants" (page 272) for the constants which you can use here.

`fData`

Variable length data, containing the address.

Discussion

The new tagged data format for addressing allows for changes in addressing formats, allowing AppleShare clients to support new addressing standards without changing the interface. The `AFPAlternateAddress` (page 182) data structure uses the `AFPTagData` structure to specify a tagged address.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

AFPVolMountInfo

Defines a volume mounting structure for an AppleShare server.

```
struct AFPVolMountInfo {
    short length;
    VolumeType media;
    short flags;
    SInt8 nbpInterval;
    SInt8 nbpCount;
    short uamType;
    short zoneNameOffset;
    short serverNameOffset;
    short volNameOffset;
    short userNameOffset;
    short userPasswordOffset;
    short volPasswordOffset;
    char AFPData[144];
};
typedef struct AFPVolMountInfo AFPVolMountInfo;
typedef AFPVolMountInfo * AFPVolMountInfoPtr;
```

Fields

`length`

The length of the `AFPVolMountInfo` structure (that is, the total length of the structure header described here plus the variable-length location data).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`media`
The volume type of the remote volume. The value `AppleShareMediaType` (a constant that translates to `'afpm'`) represents an AppleShare volume.

`flags`
If bit 0 is set, no greeting message from the server is displayed.

`nbpInterval`
The NBP retransmit interval, in units of 8 ticks.

`nbpCount`
The NBP retransmit count. This field specifies the total number of times a packet should be transmitted, including the first transmission.

`uamType`
The user authentication method used by the remote volume. AppleShare uses four methods, defined by the constants described in "Authentication Method Constants" (page 274).

`zoneNameOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare zone.

`serverNameOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare server.

`volNameOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the volume.

`userNameOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the user.

`userPasswordOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user's password (as a pascal string).

`volPasswordOffset`
The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the volume's password (as a pascal string). Some versions of the AppleShare software do not pass the information in this field to the server.

`AFPData`
The actual volume mounting information, offsets to which are contained in the preceding six fields. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the data in any order within this data field, as long as you specify the correct offsets in the offset fields.

Discussion
The only volumes that currently support the programmatic mounting functions are AppleShare servers, which use a volume mounting structure of type `AFPVolMountInfo`.

To mount an AppleShare server, fill out an `AFPVolMountInfo` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

Version Notes
AppleShare clients prior to version 3.7 mount volumes over AppleTalk only. For maximum compatibility set the `uamType` field to 1 for guest login or 3 for login using a password.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

To mount volumes using IP addresses and other address formats, use the `AFPXVolMountInfo` (page 185) structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

AFPXVolMountInfo

Defines a volume mounting structure for an AppleShare server, for AppleShare 3.7 and later.

```
struct AFPXVolMountInfo {
    short length;
    VolumeType media;
    short flags;
    SInt8 nbpInterval;
    SInt8 nbpCount;
    short uamType;
    short zoneNameOffset;
    short serverNameOffset;
    short volNameOffset;
    short userNameOffset;
    short userPasswordOffset;
    short volPasswordOffset;
    short extendedFlags;
    short uamNameOffset;
    short alternateAddressOffset;
    char AFPData[176];
};
typedef struct AFPXVolMountInfo AFPXVolMountInfo;
typedef AFPXVolMountInfo * AFPXVolMountInfoPtr;
```

Fields

`length`
: The length of the `AFPXVolMountInfo` structure (that is, the total length of the structure header described here plus the variable-length location data).

`media`
: The volume type of the remote volume. The value `AppleShareMediaType` (a constant that translates to `'afpm'`) represents an AppleShare volume.

`flags`
: Volume mount flags. See "Volume Mount Flags" (page 328) for a description of the bits in this field. In order to use the new features of the extended AFP volume mount structure, you must set the `volMountExtendedFlagsBit` bit.

`nbpInterval`
: The NBP retransmit interval, in units of 8 ticks.

`nbpCount`
: The NBP retransmit count. This field specifies the total number of times a packet should be transmitted, including the first transmission.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`uamType`
: The user authentication method used by the remote volume. AppleShare uses four methods, defined by the constants described in "Authentication Method Constants" (page 274).

`zoneNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare zone.

`serverNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the AppleShare server.

`volNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the volume.

`userNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the name (as a pascal string) of the user.

`userPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user's password (as a pascal string).

`volPasswordOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the volume's password (as a pascal string). Some versions of the AppleShare software do not pass the information in this field to the server.

`extendedFlags`
: Extended flags. See "Extended AFP Volume Mounting Information Flag" (page 289).

`uamNameOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing the user authentication module name (as a pascal string).

`alternateAddressOffset`
: The offset in bytes from the beginning of the structure to the entry in the `AFPData` field containing IP addresses, specified as a block of tagged data. This block of tagged data begins with a version byte and a count byte, followed by up to 255 tagged addresses. See `AFPAlternateAddress` (page 182).

`AFPData`
: The actual volume mounting information, offsets to which are contained in the preceding fields. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the data in any order within this data field, as long as you specify the correct offsets in the offset fields.

Discussion

To mount an AppleShare server, fill out an `AFPXVolMountInfo` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

The extended AFP volume mount information structure requires AppleShare client 3.7 and later. The new fields and flag bits allow you to specify the information needed to support TCP/IP and User Authentication Modules.

Note that, for all fields specifying an offset, if you wish to leave the string field in the `AFPData` field empty, you must specify an empty string and have the offset in the corresponding offset field point to that empty string. You cannot simply pass 0 as the offset.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

CatPositionRec

Defines a catalog position structure, which maintains the current position of a catalog search between calls to `PBCatSearchSync` or `PBCatSearchAsync`.

```
struct CatPositionRec {
    long initialize;
    short priv[6];
};
typedef struct CatPositionRec CatPositionRec;
```

Fields

`initialize`
: The starting point of the catalog search. To start searching at the beginning of a catalog, specify 0 in this field. To resume a previous search, pass the value returned by the previous call to `PBCatSearchSync` or `PBCatSearchAsync`.

`priv`
: An array of integers that is used internally by `PBCatSearchSync` and `PBCatSearchAsync`.

Discussion
When you call the `PBCatSearchSync` or `PBCatSearchAsync` function to search a volume's catalog file, you can specify, in the `ioCatPosition` field of the parameter block passed to `PBCatSearchSync` and `PBCatSearchAsync`, a catalog position structure. If a catalog search consumes more time than is allowed by the `ioSearchTime` field, `PBCatSearchSync` and `PBCatSearchAsync` store a directory-location index in that structure; when you call `PBCatSearchSync` or `PBCatSearchAsync` again, it uses that structure to resume searching where it left off.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

CInfoPBRec

Defines a catalog information parameter block for file and directory information.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
union CInfoPBRec {
    HFileInfo hFileInfo;
    DirInfo dirInfo;
};
typedef union CInfoPBRec CInfoPBRec;
typedef CInfoPBRec * CInfoPBPtr;
```

Fields

`hFileInfo`

`dirInfo`

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Files.h`

CMovePBRec

Defines a parameter block, used with the functions `PBCatMoveSync` and `PBCatMoveAsync`.

```
struct CMovePBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler1;
    StringPtr ioNewName;
    long filler2;
    long ioNewDirID;
    long filler3[2];
    long ioDirID;
};
typedef struct CMovePBRec CMovePBRec;
typedef CMovePBRec * CMovePBPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type (This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler1`
: Reserved.

`ioNewName`
: The name of the directory into which the specified file or directory is to be moved.

`filler2`
: Reserved.

`ioNewDirID`
: The directory ID of the directory into which the specified file or directory is to be moved.

`filler3`
: Reserved.

`ioDirID`
: The current directory ID of the file or directory to be moved (used in conjunction with the `ioVRefNum` and `ioNamePtr` fields).

Discussion
The low-level HFS function `PBCatMove` uses the catalog move parameter block defined by the `CMovePBRec` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

CntrlParam

Defines a parameter block used by control and status functions in the classic Device Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct CntrlParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioCRefNum;
    short csCode;
    short csParam[11];
};
typedef struct CntrlParam CntrlParam;
typedef CntrlParam * CntrlParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`ioCRefNum`

The driver reference number for the I/O operation.

`csCode`

A value identifying the type of control or status request. Each driver may interpret this number differently.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`csParam`

The control or status information passed to or from the driver. This field is declared generically as an array of eleven integers. Each driver may interpret the contents of this field differently. Refer to the driver's documentation for specific information.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Files.h`

ConstFSSpecPtr

Defines a pointer to an `FSSpec` structure.

```
typedef const FSSpec*  ConstFSSpecPtr;
```

Discussion
The only difference between "`const FSSpec*`" and the `ConstFSSpecPtr` data type is that, as a parameter, a `ConstFSSpecPtr` data type is allowed to be `NULL`. See `FSSpec` (page 226).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

ConstHFSUniStr255Param

Defines a pointer to an `HFSUniStr255` structure.

```
typedef const HFSUniStr255*  ConstHFSUniStr255Param;
```

Discussion
See `HFSUniStr255` (page 241).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

CopyParam

Defines a parameter block used by low-level HFS file copying functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct CopyParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioDstVRefNum;
    short filler8;
    StringPtr ioNewName;
    StringPtr ioCopyName;
    long ioNewDirID;
    long filler14;
    long filler15;
    long ioDirID;
};
typedef struct CopyParam CopyParam;
typedef CopyParam * CopyParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioDstVRefNum`
: A volume reference number for the destination volume.

`filler8`
: Reserved.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNewName`
A pointer to the destination pathname.

`ioCopyName`
A pointer to an optional name.

`ioNewDirID`
A destination directory ID.

`filler14`
Reserved.

`filler15`
Reserved.

`ioDirID`
A directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

CSParam

Defines a parameter block used by low-level HFS catalog search functions.

```
struct CSParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    FSSpecPtr ioMatchPtr;
    long ioReqMatchCount;
    long ioActMatchCount;
    long ioSearchBits;
    CInfoPBPtr ioSearchInfo1;
    CInfoPBPtr ioSearchInfo2;
    long ioSearchTime;
    CatPositionRec ioCatPosition;
    Ptr ioOptBuffer;
    long ioOptBufSize;
};
typedef struct CSParam CSParam;
typedef CSParam * CSParamPtr;
```

Fields
`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioMatchPtr`
A pointer to an array of `FSSpec` (page 226) structures in which the file and directory names that match the selection criteria are returned. The array must be large enough to hold the largest possible number of `FSSpec` structures, as determined by the `ioReqMatchCount` field.

`ioReqMatchCount`
The maximum number of matches to return. This number should be the number of `FSSpec` structures that will fit in the memory pointed to by the `ioMatchPtr` field. You can use this field to avoid a possible excess of matches for criteria that prove to be too general (or to limit the length of a search if the `ioSearchTime` field isn't used).

`ioActMatchCount`
The number of actual matches found.

`ioSearchBits`
The fields of the parameter blocks in the `ioSearchInfo1` and `ioSearchInfo2` fields that are relevant to the search. See "Catalog Search Bits" (page 282) for more information.

`ioSearchInfo1`
A pointer to a `CInfoPBRec` parameter block that contains the search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in the `ioSearchInfo2` field, and set the matching values in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

`ioSearchInfo2`
A pointer to a second `CInfoPBRec` parameter block that contains the search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the matching values in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioSearchTime`

A time limit on a search, in Time Manager format. Use this field to limit the run time of a single call to `PBCatSearchSync` or `PBCatSearchAsync`. A value of 0 imposes no time limit. If the value of this field is positive, it is interpreted as milliseconds. If the value of this field is negative, it is interpreted as negated microseconds.

`ioCatPosition`

A position in the catalog where searching should begin. Use this field to keep an index into the catalog when breaking down the `PBCatSearchSync` or `PBCatSearchAsync` search into a number of smaller searches. This field is valid whenever `PBCatSearchSync` or `PBCatSearchAsync` exits because it either spends the maximum time allowed by `ioSearchTime` or finds the maximum number of matches allowed by `ioReqMatchCount`.

To start at the beginning of the catalog, set the `initialize` field of `ioCatPosition` to 0. Before exiting after an interrupted search, `PBCatSearchSync` or `PBCatSearchAsync` sets that field to the next catalog entry to be searched.

To resume where the previous call stopped, pass the entire `CatPositionRec` (page 187) structure returned by the previous call as input to the next.

`ioOptBuffer`

A pointer to an optional read buffer. The `ioOptBuffer` and `ioOptBufSize` fields let you specify a part of memory as a read buffer, increasing search speed.

`ioOptBufSize`

The size of the buffer pointed to by `ioOptBuffer`. Buffer size effectiveness varies with models and configurations, but a 16 KB buffer is likely to be optimal. The size should be at least 1024 bytes and should be an integral multiple of 512 bytes.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Files.h`

DirInfo

Defines a structure which holds catalog information about a directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct DirInfo {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioACUser;
    DInfo ioDrUsrWds;
    long ioDrDirID;
    unsigned short ioDrNmFls;
    short filler3[9];
    unsigned long ioDrCrDat;
    unsigned long ioDrMdDat;
    unsigned long ioDrBkDat;
    DXInfo ioDrFndrInfo;
    long ioDrParID;
};
typedef struct DirInfo DirInfo;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
The file reference number of an open file.

`ioFVersNum`
A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
Reserved.

`ioFDirIndex`
A file and directory index. If this field contains a positive number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory having that directory index in the directory specified by the `ioVRefNum` field. (If `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

If this field contains 0, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory whose name is specified in the `ioNamePtr` field and that is located in the directory specified by the `ioVRefNum` field. (Once again, if `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

If this field contains a negative number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` ignore the `ioNamePtr` field and returns information about the directory specified in the `ioDirID` field. If both `ioDirID` and `ioVRefNum` are set to 0, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the current default directory.

`ioFlAttrib`
File or directory attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field.

`ioACUser`
The user's access rights for the specified directory. See "User Privileges Constants" (page 316) for the meaning of the bits in this field.

`ioDrUsrWds`
Information used by the Finder.

`ioDrDirID`
A directory ID. On input to `PBGetCatInfoSync` and `PBGetCatInfoAsync` , this field contains a directory ID, which is used only if the value of the `ioFDirIndex` field is negative. On output, this field contains the directory ID of the specified directory.

`ioDrNmFls`
The number of files in the directory.

`filler3`
Reserved.

`ioDrCrDat`
The date and time of the directory's creation, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the creation date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970, and that is the value which will be returned if you later try to retrieve the creation date.

Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDrMdDat`

The date and time of the last modification to the directory, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the modification date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

`ioDrBkDat`

The date and time that the directory was last backed up, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the backup date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates.

`ioDrFndrInfo`

Additional information used by the Finder.

`ioDrParID`

The directory ID of the specified directory's parent directory.

`refCon`

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Files.h`

DrvQEl

Defines a drive queue element.

```
struct DrvQEl {
    QElemPtr qLink;
    short qType;
    short dQDrive;
    short dQRefNum;
    short dQFSID;
    unsigned short dQDrvSz;
    unsigned short dQDrvSz2;
};
typedef struct DrvQEl DrvQEl;
typedef DrvQEl * DrvQElPtr;
```

Fields

`qLink`

A pointer to the next entry in the drive queue.

`qType`

Used to specify the size of the drive. If the value of this field is 0, the number of logical blocks on the drive is contained in the `dQDrvSz` field alone. If the value of this field is 1, both the `dQDrvSz` field and the `dQDrvSz2` field are used to store the number of blocks; in that case, the `dQDrvSz2` field contains the high-order word of this number and `dQDrvSz` contains the low-order word.

`dQDrive`

The drive number of the drive.

`dQRefNum`

The driver reference number of the driver controlling the device on which the volume is mounted.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`dQFSID`

An identifier for the file system handling the volume in the drive it's zero for volumes handled by the File Manager and nonzero for volumes handled by other file systems.

`dQDrvSz`

The number of logical blocks on the drive.

`dQDrvSz2`

An additional field to handle large drives. This field is only used if the `qType` field is 1.

Discussion
The File Manager maintains a list of all disk drives connected to the computer. It maintains this list in the drive queue, which is a standard operating system queue. The drive queue is initially created at system startup time. Elements are added to the queue at system startup time or when you call the `AddDrive` function. The drive queue can support any number of drives, limited only by memory space. Each element in the drive queue contains information about the corresponding drive.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

DTPBRec

Defines the desktop database parameter block used by the desktop database functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct DTPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioDTRefNum;
    short ioIndex;
    long ioTagInfo;
    Ptr ioDTBuffer;
    long ioDTReqCount;
    long ioDTActCount;
    SInt8 ioFiller1;
    UInt8 ioIconType;
    short ioFiller2;
    long ioDirID;
    OSType ioFileCreator;
    OSType ioFileType;
    long ioFiller3;
    long ioDTLgLen;
    long ioDTPyLen;
    short ioFiller4[14];
    long ioAPPLParID;
};
typedef struct DTPBRec DTPBRec;
typedef DTPBRec * DTPBPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
A pointer to a file, directory, or volume name. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
The volume reference number.

`ioDTRefNum`
The desktop database reference number.

`ioIndex`
The index into icon list.

`ioTagInfo`
The tag information.

`ioDTBuffer`
The data buffer.

`ioDTReqCount`
The requested length of data.

`ioDTActCount`
The actual length of data.

`ioFiller1`
Unused.

`ioIconType`
The icon type.

`ioFiller2`
Unused.

`ioDirID`
The parent directory ID.

`ioFileCreator`
The file creator.

`ioFileType`
The file type.

`ioFiller3`
Unused.

`ioDTLgLen`
The logical length of the desktop database.

`ioDTPyLen`
The physical length of the desktop database.

`ioFiller4`
Unused.

`ioAPPLParID`
The parent directory ID of an application.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

FCBPBRec

Defines a file control block (FCB) parameter block used by the functions `PBGetFCBInfoSync` and `PBGetFCBInfoAsync`.

```
struct FCBPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    short filler;
    short ioFCBIndx;
    short filler1;
    long ioFCBFlNm;
    short ioFCBFlags;
    unsigned short ioFCBStBlk;
    long ioFCBEOF;
    long ioFCBPLen;
    long ioFCBCrPs;
    short ioFCBVRefNum;
    long ioFCBClpSiz;
    long ioFCBParID;
};
typedef struct FCBPBRec FCBPBRec;
typedef FCBPBRec * FCBPBPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
: The file reference number of an open file.

`filler`
: Reserved.

`ioFCBIndx`
: An index for use with the `PBGetFCBInfoSync` and `PBGetFCBInfoAsync` functions.

`filler1`
: Reserved.

`ioFCBFlNm`
: The file ID.

`ioFCBFlags`
: Flags describing the status of the file. See "FCB Flags" (page 293) for the meanings of the bits in this field.

`ioFCBStBlk`
: The number of the first allocation block of the file.

`ioFCBEOF`
: The logical length (logical end-of-file) of the file.

`ioFCBPLen`
: The physical length (physical end-of-file) of the file.

`ioFCBCrPs`
: The current position of the file mark.

`ioFCBVRefNum`
: The volume reference number.

`ioFCBClpSiz`
: The file clump size.

`ioFCBParID`
: The file's parent directory ID.

Discussion

The low-level HFS function `PBGetFCBInfo` uses the file control block parameter block defined by the `FCBPBRec` data type.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

FIDParam

Defines a parameter block used by low-level HFS file ID functions.

```
struct FIDParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler14;
    StringPtr ioDestNamePtr;
    long filler15;
    long ioDestDirID;
    long filler16;
    long filler17;
    long ioSrcDirID;
    short filler18;
    long ioFileID;
};
typedef struct FIDParam FIDParam;
typedef FIDParam * FIDParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler14`
: Reserved.

`ioDestNamePtr`
: A pointer to the name of the destination file.

`filler15`
: Reserved.

`ioDestDirID`
: The parent directory ID of the destination file.

`filler16`
: Reserved.

`filler17`
: Reserved.

`ioSrcDirID`
: The parent directory ID of the source file.

`filler18`
: Reserved.

`ioFileID`
: The file ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FileParam

Defines a parameter block used by low-level functions for getting and setting file information.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FileParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioFlVersNum;
    FInfo ioFlFndrInfo;
    unsigned long ioFlNum;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
};
typedef struct FileParam FileParam;
typedef FileParam * FileParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
The file reference number of an open file.

`ioFVersNum`
A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
Reserved.

`ioFDirIndex`
A directory index for use with the `PBHGetFInfoSync` (page 444) and `PBHGetFInfoAsync` (page 442) functions.

`ioFlAttrib`
File attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field.

`ioFlVersNum`
A file version number. This feature is no longer supported, and you must always set this field to 0.

`ioFlFndrInfo`
Information used by the Finder.

`ioFlNum`
A file ID.

`ioFlStBlk`
The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`
The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`
The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`
The first allocation block of the resource fork. This field contains 0 if the file's resource fork is empty.

`ioFlRLgLen`
The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
The date and time of the file's creation, specified in seconds since midnight, January 1, 1904.

`ioFlMdDat`
The date and time of the last modification to the file, specified in seconds since midnight, January 1, 1904.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FNSubscriptionRef

```
typedef struct OpaqueFNSubscriptionRef * FNSubscriptionRef;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

FNSubscriptionUPP

```
typedef FNSubscriptionProcPtr FNSubscriptionUPP;
```

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Files.h`

ForeignPrivParam

Defines a parameter block used by low-level HFS foreign privileges functions.

```
struct ForeignPrivParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long ioFiller21;
    long ioFiller22;
    Ptr ioForeignPrivBuffer;
    long ioForeignPrivActCount;
    long ioForeignPrivReqCount;
    long ioFiller23;
    long ioForeignPrivDirID;
    long ioForeignPrivInfo1;
    long ioForeignPrivInfo2;
    long ioForeignPrivInfo3;
    long ioForeignPrivInfo4;
};
typedef struct ForeignPrivParam ForeignPrivParam;
typedef ForeignPrivParam * ForeignPrivParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFiller21`
Reserved.

`ioFiller22`
Reserved.

`ioForeignPrivBuffer`
A pointer to a buffer containing access-control information about the foreign file system.

`ioForeignPrivActCount`
The size of the buffer pointed to by the `ioForeignPrivBuffer` field.

`ioForeignPrivReqCount`
The amount of the buffer pointed to by the `ioForeignPrivBuffer` field that was actually used to hold data.

`ioFiller23`
Reserved.

`ioForeignPrivDirID`
The parent directory ID of the foreign file or directory.

`ioForeignPrivInfo1`
A long word that may contain privileges data.

`ioForeignPrivInfo2`
A long word that may contain privileges data.

`ioForeignPrivInfo3`
A long word that may contain privileges data.

`ioForeignPrivInfo4`
A long word that may contain privileges data.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

FSCatalogBulkParam

Defines a parameter block used to retrieve catalog information in bulk on HFS Plus volumes.

```
struct FSCatalogBulkParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    Boolean containerChanged;
    UInt8 reserved;
    FSIteratorFlags iteratorFlags;
    FSIterator iterator;
    const FSRef * container;
    ItemCount maximumItems;
    ItemCount actualItems;
    FSCatalogInfoBitmap whichInfo;
    FSCatalogInfo * catalogInfo;
    FSRef * refs;
    FSSpec * specs;
    HFSUniStr255 * names;
    const FSSearchParams * searchParams;
};
typedef struct FSCatalogBulkParam FSCatalogBulkParam;
typedef FSCatalogBulkParam * FSCatalogBulkParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`containerChanged`
: A Boolean value indicating whether or not the container has changed since the last call to `PBGetCatalogInfoBulkSync` or `PBGetCatalogInfoBulkAsync`.

`reserved`
: Reserved.

`iteratorFlags`
: A set of flags which specifies how the iterator should iterate over the container. See "Iterator Flags" (page 310) for the meaning of the constants used here.

`iterator`
: A catalog iterator.

`container`
: An `FSRef` for the directory or volume to iterate over.

`maximumItems`
: The maximum number of items to return information about.

`actualItems`
: The actual number of items returned.

`whichInfo`
: A bitmap indicating which fields of the catalog information structure to return. See "Catalog Information Bitmap Constants" (page 277) for the bits defined for this field.

`catalogInfo`
: A pointer to an array of catalog information structures. On input, you should pass a pointer to an array of `maximumItemsFSCatalogInfo` (page 212) structures. On return, `actualItems` structures will be filled out with the information requested in the `whichInfo` field. If you do not wish any catalog information to be returned, pass a `NULL` pointer in this field and pass the constant `kFSCatInfoNone` in the `whichInfo` field.

`refs`
: A pointer to an array of `FSRef` structures. On input, you should pass a pointer to `maximumItemsFSRef` structures. On return, `actualItems` structures will be filled out. If you do not wish any `FSRef` structures to be returned, pass a `NULL` pointer in this field.

`specs`
: A pointer to an array of `FSSpec` structures. On input, you should pass a pointer to `maximumItems` file system specifications. On return, `actualItemsFSSpec` structures will be filled in. If you do not wish any `FSSpec` information to be returned, pass a `NULL` pointer in this field.

`names`
: A pointer to an array of Unicode names. On input, you should pass a pointer to an array of `maximumItemsHFSUniStr255` structures. On return, `actualItems` structures will contain Unicode names. If you do not wish any file or directory names to be returned, pass a `NULL` pointer in this field.

`searchParams`
: A pointer to an `FSSearchParams` (page 225) structure, specifying the values to match against.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

FSCatalogInfo

Holds basic information about a file or directory.

```
struct FSCatalogInfo {
    UInt16 nodeFlags;
    FSVolumeRefNum volume;
    UInt32 parentDirID;
    UInt32 nodeID;
    UInt8 sharingFlags;
    UInt8 userPrivileges;
    UInt8 reserved1;
    UInt8 reserved2;
    UTCDateTime createDate;
    UTCDateTime contentModDate;
    UTCDateTime attributeModDate;
    UTCDateTime accessDate;
    UTCDateTime backupDate;
    UInt32 permissions[4];
    UInt8 finderInfo[16];
    UInt8 extFinderInfo[16];
    UInt64 dataLogicalSize;
    UInt64 dataPhysicalSize;
    UInt64 rsrcLogicalSize;
    UInt64 rsrcPhysicalSize;
    UInt32 valence;
    TextEncoding textEncodingHint;
};
typedef struct FSCatalogInfo FSCatalogInfo;
typedef FSCatalogInfo * FSCatalogInfoPtr;
```

Fields

`nodeFlags`
Node flags. This field has two defined bits that indicate whether an object is a file or folder, and whether a file is locked (constants `kFSNodeIsDirectoryMask` and `kFSNodeLockedMask`). See "Catalog Information Node Flags" (page 280) for the values you can use here.

`volume`
The object's volume reference.

`parentDirID`
The ID of the directory that contains the given object. The root directory of a volume always has ID `fsRtDirID` (2); the parent of the root directory is ID `fsRtParID` (1). Note that there is no object with ID `fsRtParID`; this is merely used when the File Manager is asked for the parent of the root directory.

`nodeID`
The file or directory ID.

`sharingFlags`
The object's sharing flags. See "Catalog Information Sharing Flags " (page 282) for the meaning of the bits defined for this field.

`userPrivileges`
The user's effective AFP privileges (same as `ioACUser` in the old `HFileInfo` and `DirInfo` structures). See "User Privileges Constants" (page 316).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`reserved1`
Reserved.

`reserved2`
Reserved.

`createDate`
The date and time of the creation of the object. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, `FSGetCatalogInfo`, `PBGetCatalogInfoSync`, and `PBGetCatalogInfoAsync` return 0 in this field.

`contentModDate`
The date and time that the data or resource fork was last modified.

`attributeModDate`
The date and time that an attribute of the object (such as a fork other than the data or resource fork) was last modified.

`accessDate`
The date and time that the object was last accessed. The Mac OS 9 File Manager does not automatically update the `accessDate` field; it exists primarily for use by other operating systems (notably Mac OS X).

`backupDate`
The date and time of the object's last backup. This field is not updated by the File Manager a backup utility may use this field if it wishes. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, `FSGetCatalogInfo`, `PBGetCatalogInfoSync`, and `PBGetCatalogInfoAsync` return 0 in this field.

`permissions`
User and group permission information. The Mac OS 8 and 9 File Manager does not use or enforce this permission information. It could be used by a file server program or other operating system (primarily Mac OS X). In Mac OS X, this array contains the file system permissions of the returned item. To use this information, coerce the parameter to a `FSPermissionInfo` (page 222) structure.

`finderInfo`
Basic Finder information for the object. This information is available in the catalog information, instead of in a named fork, for historical reasons. The File Manager does not interpret the contents of these fields. To use this information, coerce the parameter to a `FileInfo` or `FolderInfo` structure.

`extFinderInfo`
Extended Finder information for the object. This information is available in the catalog information, instead of in a named fork, for historical reasons. The File Manager does not interpret the contents of these fields. To use this information, coerce the parameter to an `ExtendedFileInfo` or `ExtendedFolderInfo` structure.

`dataLogicalSize`
The size of the data fork in bytes (the fork's logical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`dataPhysicalSize`
The amount of disk space, in bytes, occupied by the data fork (the fork's physical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`rsrcLogicalSize`
The size of the resource fork (the fork's logical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`rsrcPhysicalSize`

The amount of disk space occupied by the resource fork (the fork's physical size). The information in this field is only valid for files do not rely upon the value returned in this field for folders.

`valence`

For folders only, the number of items (files plus directories) contained within the directory. For files, it is set to zero. Many volume formats do not store a field containing a directory's valence. For those volume formats, this field is very expensive to compute. Think carefully before you ask the File Manager to return this field.

`textEncodingHint`

The `textEncodingHint` field is used in conjunction with the Unicode filename of the object. It is an optional hint that can be used by the volume format when converting the Unicode to some other encoding. For example, HFS Plus stores this value and uses it when converting the name to a Mac OS encoding, such as when the name is returned by `PBGetCatInfoSync` or `PBGetCatInfoAsync`. As another example, HFS volumes use this value to convert the Unicode name to a Mac OS encoded name stored on disk. If the entire Unicode name can be converted to a single Mac OS encoding, then that encoding should be used as the `textEncodingHint`; otherwise, a text encoding corresponding to the first characters of the name will probably provide the best user experience.

If a `textEncodingHint` is not supplied when a file or directory is created or renamed, the volume format will use a default value. This default value may not be the best possible choice for the given filename. Whenever possible, a client should supply a `textEncodingHint`.

Discussion
The `FSCatalogInfoBitmap` type is used to indicate which fields of the `FSCatalogInfo` should be set or retrieved. If the bit corresponding to a particular field is not set, then that field is not changed if the `FSCatalogInfo` is an output parameter, and that field is ignored if the `FSCatalogInfo` is an input parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSCatalogInfoBitmap

Describes which fields of the `FSCatalogInfo` structure you wish to retrieve or set.

```
typedef UInt32 FSCatalogInfoBitmap;
```

Discussion
If the bit corresponding to a particular field is not set in the bitmap, then that field is not changed in the `FSCatalogInfo` structure if it is an output parameter, and that field is ignored if the `FSCatalogInfo` structure is an input parameter. See "Catalog Information Bitmap Constants" (page 277) for a description of the constants you should use with this data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSEjectStatus

```
typedef UInt32 FSEjectStatus;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSFileOperationClientContext

Specifies user-defined data and callbacks associated with an asynchronous file operation.

```
struct FSFileOperationClientContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct FSFileOperationClientContext FSFileOperationClientContext;
```

Fields

`version`
: The version number of the structure; this field should always contain 0.

`info`
: A generic pointer to your user-defined data. This pointer is passed back to your application when you check the status of the file operation. There are two ways you can ask the File Manager for status information about a file operation: by supplying a status callback function when you start the operation, or by calling a file operation status function directly.

`retain`
: An optional callback function that the File Manager can use to retain the user-defined data specified in the `info` parameter. If your data is a Core Foundation object, you can simply specify the function `CFRetain`. If no callback is needed, set this field to `NULL`.

`release`
: An optional callback function that the File Manager can use to release the user-defined data specified in the `info` parameter. If your data is a Core Foundation object, you can simply specify the function `CFRelease`. If no callback is needed, set this field to `NULL`.

`copyDescription`
: An optional callback function that the File Manager can use to create a descriptive string representation of your user-defined data for debugging purposes. If no callback is needed, set this field to `NULL`.

Discussion
You supply a client context when calling functions such as `FSCopyObjectAsync` (page 50) or `FSMoveObjectAsync` (page 83) that start an asynchronous copy or move operation.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSFileOperationRef

Defines an opaque type that represents an asynchronous file operation.

```
typedef struct __FSFileOperation * FSFileOperationRef;
```

Discussion
You supply a file operation object when calling functions such as `FSCopyObjectAsync` (page 50) or `FSMoveObjectAsync` (page 83) to start an asynchronous copy or move operation. You can also use a file operation object to check the status of a file operation or to cancel the operation.

To perform an asynchronous file operation:

1. Create a file operation object using the function `FSFileOperationCreate` (page 61).
2. Pass the object to the function `FSFileOperationScheduleWithRunLoop` (page 62) to schedule the operation.
3. Pass the object to one of the asynchronous file operation functions to start the operation.

The `FSFileOperationRef` opaque type is a standard Core Foundation data type. It is derived from CFType and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

FSForkCBInfoParam

Defines a parameter block used by low-level HFS Plus fork control block functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FSForkCBInfoParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    SInt16 desiredRefNum;
    SInt16 volumeRefNum;
    SInt16 iterator;
    SInt16 actualRefNum;
    FSRef * ref;
    FSForkInfo * forkInfo;
    HFSUniStr255 * forkName;
};
typedef struct FSForkCBInfoParam FSForkCBInfoParam;
typedef FSForkCBInfoParam * FSForkCBInfoParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`desiredRefNum`
: A fork reference number.

`volumeRefNum`
: The volume reference number of the volume to match, or zero to match all volumes.

`iterator`
: An iterator. Set to zero to start iteration.

`actualRefNum`
: On return, the actual fork reference number found.

`ref`
: A pointer to an `FSRef` for the specified fork.

`forkInfo`
: A pointer to a fork information structure, `FSForkInfo` (page 218).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`forkName`
A pointer to the fork's Unicode name.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSForkInfo

Contains information about an open fork.

```
struct FSForkInfo {
    SInt8 flags;
    SInt8 permissions;
    FSVolumeRefNum volume;
    UInt32 reserved2;
    UInt32 nodeID;
    UInt32 forkID;
    UInt64 currentPosition;
    UInt64 logicalEOF;
    UInt64 physicalEOF;
    UInt64 process;
};
typedef struct FSForkInfo FSForkInfo;
typedef FSForkInfo * FSForkInfoPtr;
```

Fields

`flags`
Flags describing the status of the fork. See "FCB Flags" (page 293) for a description of the bits in this field.

`permissions`
User and group permission information.

`volume`
A volume specification. This can be a volume reference number, drive number, or 0 for the default volume.

`reserved2`
Reserved.

`nodeID`
The file or directory ID of the file or directory with which the fork is associated.

`forkID`
The fork ID.

`currentPosition`
The current position within the fork.

`logicalEOF`
The logical size of the fork.

`physicalEOF`
The physical size of the fork.

`process`
The process which opened the fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
This data type is used in the `forkInfo` parameter of the `FSGetForkCBInfo` function, and in the `forkInfo` field of the `FSForkCBInfoParam` parameter block passed to the `PBGetForkCBInfoSync` and `PBGetForkCBInfoAsync` functions. When these functions return, the `FSForkInfo` structure contains information about the specified open fork.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSForkIOParam

Defines a parameter block used by low-level HFS Plus fork I/O functions.

```
struct FSForkIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    void * reserved1;
    SInt16 reserved2;
    SInt16 forkRefNum;
    UInt8 reserved3;
    SInt8 permissions;
    const FSRef * ref;
    Ptr buffer;
    UInt32 requestCount;
    UInt32 actualCount;
    UInt16 positionMode;
    SInt64 positionOffset;
    FSAllocationFlags allocationFlags;
    UInt64 allocationAmount;
    UniCharCount forkNameLength;
    const UniChar * forkName;
    CatPositionRec forkIterator;
    HFSUniStr255 * outForkName;
};
typedef struct FSForkIOParam FSForkIOParam;
typedef FSForkIOParam * FSForkIOParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`reserved1`
: Reserved.

`reserved2`
: Reserved.

`forkRefNum`
: A reference number for a fork.

`reserved3`
: Reserved.

`permissions`
: The desired type of access to the specified fork. See "File Access Permission Constants" (page 295) for a description of the types of access that you can request.

`ref`
: An `FSRef` for the file or directory to open.

`buffer`
: A pointer to a data buffer.

`requestCount`
: The number of bytes requested for the given operation.

`actualCount`
: The actual number of bytes completed by the call.

`positionMode`
: A constant indicating the base location within the file for the start of the operation. See "Position Mode Constants" (page 314) for the meaning of the constants you can use in this field.

`positionOffset`
: The offset from the base location specified in the `positionMode` offset for the start of the operation.

`allocationFlags`
: A set of bit flags used by the `FSAllocateFork` (page 43) function to control how space is allocated. See "Allocation Flags" (page 273) for a description of the defined flags.

`allocationAmount`
: For the `FSAllocateFork` (page 43) function, the amount of space, in bytes, to allocate.

`forkNameLength`
: The length of the file or directory name passed in the `forkName` field, in Unicode characters.

`forkName`
: A pointer to the file or directory's Unicode name. This field is an input parameter functions which return the file or directory name in the parameter block use the `outForkName` field.

`forkIterator`
: A fork iterator.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`outForkName`

A pointer to the file or directory's Unicode name this is an output parameter. For functions which require the file or directory name as an input argument, you should pass a pointer to that name in the `forkName` field and pass the length of the name in the `forkNameLength` field.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSIterator

Refers to a position within the catalog, used when iterating over files and folders in a directory.

```
typedef struct OpaqueFSIterator * FSIterator;
```

Discussion
This data type is like a file reference number because it maintains state internally to the File Manager and must be explicitly opened and closed.

An `FSIterator` is returned by `FSOpenIterator` and is passed as input to `FSGetCatalogInfoBulk`, `FSCatalogSearch` and `FSCloseIterator`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSMountStatus

```
typedef UInt32 FSMountStatus;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSPermissionInfo

```
struct FSPermissionInfo {
    UInt32 userID;
    UInt32 groupID;
    UInt8 reserved1;
    UInt8 userAccess;
    UInt16 mode;
    UInt32 reserved2;
};
typedef struct FSPermissionInfo FSPermissionInfo;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSRangeLockParam

Defines a parameter block for use with 64-bit range locking functions.

```
struct FSRangeLockParam {
    QElemPtr qLink;
    SInt16 qType;
    SInt16 ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    FSIORefNum forkRefNum;
    UInt64 requestCount;
    UInt16 positionMode;
    SInt64 positionOffset;
    UInt64 rangeStart;
};
typedef struct FSRangeLockParam FSRangeLockParam;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

FSRangeLockParamPtr

Defines a pointer to a range lock parameter block.

```
typedef FSRangeLockParam *FSRangeLockParamPtr;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSRef

Identifies a directory or file, including a volume's root directory.

```
struct FSRef {
    UInt8 hidden[80];
};
typedef struct FSRef FSRef;
typedef FSRef * FSRefPtr;
```

Discussion
This data type's purpose is similar to an `FSSpec` except that an `FSRef` is completely opaque. An `FSRef` contains whatever information is needed to find the given object; the internal structure of an `FSRef` is likely to vary based on the volume format, and may vary based on the particular object being identified.

The client of the File Manager cannot examine the contents of an `FSRef` to extract information about the parent directory or the object's name. Similarly, an `FSRef` cannot be constructed directly by the client; the `FSRef` must be constructed and returned via the File Manager. There is no need to call the File Manager to dispose an `FSRef`.

To determine the volume, parent directory and name associated with an `FSRef`, or to get an equivalent `FSSpec`, use the `FSGetCatalogInfo` call.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSRefParam

Defines a parameter block used by low-level HFS Plus functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FSRefParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    ConstStringPtr ioNamePtr;
    short ioVRefNum;
    SInt16 reserved1;
    UInt8 reserved2;
    UInt8 reserved3;
    const FSRef * ref;
    FSCatalogInfoBitmap whichInfo;
    FSCatalogInfo * catInfo;
    UniCharCount nameLength;
    const UniChar * name;
    long ioDirID;
    FSSpec * spec;
    FSRef * parentRef;
    FSRef * newRef;
    TextEncoding textEncodingHint;
    HFSUniStr255 * outName;
};
typedef struct FSRefParam FSRefParam;
typedef FSRefParam * FSRefParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—you should set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
A volume reference number, or 0 for the default volume.

`reserved1`
Reserved.

`reserved2`
Reserved.

`reserved3`
Reserved.

`ref`
The `FSRef` describing the file or directory which is the target of the call.

`whichInfo`
An `FSCatalogInfoBitmap` which describes the fields of the catalog information structure passed in the `catInfo` field which are to be retrieved or set.

`catInfo`
A catalog information structure containing information about the specified file or directory.

`nameLength`
The length of the file or directory's name, for the `PBCreateSync`, `PBCreateAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`name`
A pointer to the file or directory's Unicode name, for the `PBCreateSync`, `PBCreateAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`ioDirID`
The directory ID of the specified file or directory's parent directory.

`spec`
The target or source `FSRef`.

`parentRef`
The secondary or destination `FSRef`. (Or the ref of the directory to move another file or directory to).

`newRef`
The output `FSRef` (ie, a new `FSRef`).

`textEncodingHint`
A text encoding hint for the file or directory's Unicode name, used by the `PBMakeFSRefSync`, `PBMakeFSRefAsync`, `PBRenameSync`, and `PBRenameAsync` functions.

`outName`
On output, a pointer to the Unicode name of the file or directory, used by the `PBGetCatalogInfoSync` and `PBGetCatalogInfoAsync` functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSSearchParams

Describes the search criteria for a catalog information search.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FSSearchParams {
    Duration searchTime;
    OptionBits searchBits;
    UniCharCount searchNameLength;
    const UniChar * searchName;
    FSCatalogInfo * searchInfo1;
    FSCatalogInfo * searchInfo2;
};
typedef struct FSSearchParams FSSearchParams;
typedef FSSearchParams * FSSearchParamsPtr;
```

Fields

`searchTime`

A Time Manager duration for the duration of the search. If you specify a non-zero value in this field, the search may terminate after the specified time, even if the maximum number of requested objects has not been returned and the entire catalog has not been scanned.

If this value is negative, the time is interpreted in microseconds; if positive, it is interpreted as milliseconds. If `searchTime` is zero, there is no time limit on the search.

`searchBits`

A set of bits specifying which catalog information fields to search on. See "Catalog Search Constants" (page 285) for the constants which you can use here.

`searchNameLength`

The length of the Unicode name to search by.

`searchName`

A pointer to the Unicode name to search by.

`searchInfo1`

An `FSCatalogInfo` (page 212) structure which specifies the values and lower bounds of a search.

`searchInfo2`

A `FSCatalogInfo` (page 212) structure which specifies the masks and upper bounds of a search.

Discussion

Used by `FSCatalogSearch`, `PBCatalogSearchSync`, and `PBCatalogSearchAsync` to specify the criteria for a catalog search.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSSpec

Specifies the name and location of a file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FSSpec {
    short vRefNum;
    long parID;
    StrFileName name;
};
typedef struct FSSpec FSSpec;
typedef FSSpec * FSSpecPtr;
```

Fields

`vRefNum`

The volume reference number of the volume containing the specified file or directory.

`parID`

The parent directory ID of the specified file or directory (the directory ID of the directory containing the given file or directory).

`name`

The name of the specified file or directory. In Carbon, this name must be a leaf name; the name cannot contain a semicolon.

Discussion

The `FSSpec` structure can describe only a file or a directory, not a volume. A volume can be identified by its root directory, although the system software never uses an `FSSpec` structure to describe a volume. The directory ID of the root's parent directory is `fsRtParID`. The name of the root directory is the same as the name of the volume.

If you need to convert a file specification into an `FSSpec` structure, call the function `FSMakeFSSpec` (page 346) . Do not fill in the fields of an `FSSpec` structure yourself.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

FSSpecArrayPtr

Defines a pointer to an array of `FSSpec` structures.

```
typedef FSSpecPtr FSSpecArrayPtr;
```

Discussion

See `FSSpec` (page 226).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSUnmountStatus

```
typedef UInt32 FSUnmountStatus;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSVolumeEjectUPP

```
typedef FSVolumeEjectProcPtr FSVolumeEjectUPP;
```

Discussion
For more information, see the description of the `FSVolumeEjectProcPtr` (page 177) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSVolumeInfo

Used when getting or setting information about a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct FSVolumeInfo {
    UTCDateTime createDate;
    UTCDateTime modifyDate;
    UTCDateTime backupDate;
    UTCDateTime checkedDate;
    UInt32 fileCount;
    UInt32 folderCount;
    UInt64 totalBytes;
    UInt64 freeBytes;
    UInt32 blockSize;
    UInt32 totalBlocks;
    UInt32 freeBlocks;
    UInt32 nextAllocation;
    UInt32 rsrcClumpSize;
    UInt32 dataClumpSize;
    UInt32 nextCatalogID;
    UInt8 finderInfo[32];
    UInt16 flags;
    UInt16 filesystemID;
    UInt16 signature;
    UInt16 driveNumber;
    short driverRefNum;
};
typedef struct FSVolumeInfo FSVolumeInfo;
typedef FSVolumeInfo * FSVolumeInfoPtr;
```

Fields

`createDate`
: The date and time the volume was created. A value of 0 means that the volume creation date is unknown.

`modifyDate`
: The last time when the volume was modified in any way. A value of 0 means “never” or “unknown.

`backupDate`
: Indicates when the volume was last backed up. This field is for use by backup utilities. A value of 0 means “never” or “unknown.

`checkedDate`
: The last date and time that the volume was checked for consistency. A value of 0 means “never” or “unknown.

`fileCount`
: The total number of files on the volume, or 0 if unknown.

`folderCount`
: The total number of folders on the volume, or 0 if unknown. Note that no root directory counts.

`totalBytes`
: The size of the volume in bytes.

`freeBytes`
: The number of bytes of free space on the volume.

`blockSize`
: The size of an allocation block, in bytes. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`totalBlocks`
: The total number of allocation blocks on the volume. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`freeBlocks`
: The number of unused allocation blocks on the volume. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`nextAllocation`
: A hint for where to start searching for free space during an allocation. This field is only appropriate for volume formats (such as HFS and HFS Plus) that allocate space in fixed-size pieces; other volume formats may not have a similar concept, and may set this field to zero.

`rsrcClumpSize`
: Default resource fork clump size. When a fork is automatically grown as it is written, the File Manager attempts to allocate space that is a multiple of the clump size. This field is zero for volume formats that don't support the notion of a clump size.

`dataClumpSize`
: Default data fork clump size. When a fork is automatically grown as it is written, the File Manager attempts to allocate space that is a multiple of the clump size. This field is zero for volume formats that don't support the notion of a clump size.

`nextCatalogID`
: The next unused catalog node ID. Some volume formats (such as HFS and HFS Plus) use a monotonically increasing number for the catalog node ID (i.e. File ID or Directory ID) of newly created files and directories. For those volume formats, the `nextCatalogID` is the next file/directory ID that will be assigned. For other volume formats, this field will be zero.

`finderInfo`
: Information used by Finder, such as the Directory ID of the System Folder. Some volume formats do not support Finder information for a volume and will set this field to all zeroes.

`flags`
: This field contains bit flags holding information about the volume. See "Volume Information Flags" (page 326) for the attribute constants you can use here.

`filesystemID`
: Identifies the filesystem implementation that is handling the volume; this is zero for HFS and HFS Plus volumes.

`signature`
: This field is used to distinguish between volume formats supported by a single filesystem implementation.

`driveNumber`
: The drive number for the drive (drive queue element) associated with the volume. Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

`driverRefNum`
: The driver reference number for the drive (drive queue element) associated with the volume.

Discussion
This structure contains information about a volume as a whole information about a volume's root directory would use the `FSCatalogInfo` (page 212) structure.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

FSVolumeInfoBitmap

Describes which fields of the `FSVolumeInfo` structure you wish to retrieve or set.

```
typedef UInt32 FSVolumeInfoBitmap;
```

Discussion
If the bit corresponding to a particular field is not set in the bitmap, then that field is not changed in the `FSVolumeInfo` structure if it is an output parameter, and that field is ignored if the `FSVolumeInfo` structure is an input parameter. See "Volume Information Bitmap Constants" (page 324) for a description of the constants you should use with this data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSVolumeInfoParam

Defines a parameter block used by low-level HFS Plus volume manipulation functions.

```
struct FSVolumeInfoParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    FSVolumeRefNum ioVRefNum;
    UInt32 volumeIndex;
    FSVolumeInfoBitmap whichInfo;
    FSVolumeInfo * volumeInfo;
    HFSUniStr255 * volumeName;
    FSRef * ref;
};
typedef struct FSVolumeInfoParam FSVolumeInfoParam;
typedef FSVolumeInfoParam * FSVolumeInfoParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a volume name. This field is unused.

`ioVRefNum`
: The volume reference number.

`volumeIndex`
: The volume index. If this field is 0, the value in the `ioVRefNum` field only is used to identify the target volume.

`whichInfo`
: A bitmap indicating which volume information fields to retrieve or set in the `FSVolumeInfo` (page 228) structure passed in the `volumeInfo` field. See "Volume Information Bitmap Constants" (page 324) for the meaning of the bits in this field.

`volumeInfo`
: A pointer to a volume information structure containing the requested volume information on return, or the new values of the volume information to set on input. See `FSVolumeInfo` (page 228).

`volumeName`
: On output, a pointer to the volume's name.

`ref`
: A pointer to an `FSRef` for the specified volume's root directory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSVolumeMountUPP

```
typedef FSVolumeMountProcPtr FSVolumeMountUPP;
```

Discussion
For more information, see the description of the `FSVolumeMountProcPtr` (page 178) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSVolumeOperation

```
typedef struct OpaqueFSVolumeOperation * FSVolumeOperation;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

FSVolumeRefNum

Identifies a particular mounted volume.

```
typedef SInt16 FSVolumeRefNum;
```

Discussion
This data type is the same as the 16-bit volume refnum previously passed in the `ioVRefNum` fields of a parameter block; this is simply a new type name for the old data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

FSVolumeUnmountUPP

```
typedef FSVolumeUnmountProcPtr FSVolumeUnmountUPP;
```

Discussion
For more information, see the description of the `FSVolumeUnmountProcPtr` (page 179) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Files.h`

GetVolParmsInfoBuffer

Defines a volume attributes buffer, used by the `PBHGetVolParmsSync` and `PBHGetVolParmAsync` functions to return volume information.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct GetVolParmsInfoBuffer {
    short vMVersion;
    long vMAttrib;
    Handle vMLocalHand;
    long vMServerAdr;
    long vMVolumeGrade;
    short vMForeignPrivID;
    long vMExtendedAttributes;
    void * vMDeviceID;
    UniCharCount vMMaxNameLength;
};
typedef struct GetVolParmsInfoBuffer GetVolParmsInfoBuffer;
```

Fields

`vMVersion`

The version number of the attributes buffer structure. Currently this field returns 1, 2, 3 or 4. Version 3 is introduced to support the HFS Plus APIs.

`vMAttrib`

A 32-bit quantity that encodes information about the volume attributes. See "Volume Attribute Constants" (page 317) for the meaning of the bits in this field.

`vMLocalHand`

A handle to private data for shared volumes. On creation of the VCB (right after mounting), this field is a handle to a 2-byte block of memory. The Finder uses this for its local window list storage, allocating and deallocating memory as needed. It is disposed of when the volume is unmounted. Your application should treat this field as reserved.

`vMServerAdr`

For AppleTalk server volumes, this field contains the internet address of an AppleTalk server volume. Your application can inspect this field to tell which volumes belong to which server; the value of this field is 0 if the volume does not have a server.

`vMVolumeGrade`

The relative speed rating of the volume. The scale used to determine these values is currently uncalibrated. In general, lower values indicate faster speeds. A value of 0 indicates that the volume's speed is unrated. The buffer version returned in the `vMVersion` field must be greater than 1 for this field to be meaningful.

`vMForeignPrivID`

An integer representing the privilege model supported by the volume. Currently two values are defined for this field: 0 represents a standard HFS or HFS Plus volume that might or might not support the AFP privilege model; `fsUnixPriv` represents a volume that supports the A/UX privilege model. The buffer version returned in the `vMVersion` field must be greater than 1 for this field to be meaningful.

`vMExtendedAttributes`

Contains bits that describe a volume's extended attributes. For this field to be meaningful, the `vMVersion` must be greater than 2. See "Extended Volume Attributes" (page 289) for the meaning of the bits in this field.

`vMDeviceID`

A device name identifying the device in /dev that corresponds to the volume. You can use this string to build a POSIX path to the device for use with IOKit APIs.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`vMMaxNameLength`

Discussion
Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer`. Volumes that don't implement the HFS Plus APIs may still implement version 3 of the `GetVolParmsInfoBuffer`. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear (zero), then the volume does not implement the HFS Plus APIs, and they are being emulated for that volume by the File Manager itself.

If a volume does not implement the HFS Plus APIs, and supports version 2 or earlier of the `GetVolParmsInfoBuffer`, it cannot itself describe whether it supports the `FSCatalogSearch` (page 45) or `FSExchangeObjects` calls. The compatibility layer will implement the `FSCatalogSearch` call if the volume supports the `PBCatSearch` call (i.e. the `bHasCatSearch` bit of `vMAttrib` is set). The compatibility layer will implement the `FSExchangeObjects` call if the volume supports `PBExchangeFiles` (i.e. the `bHasFileIDs` bit of `vMAttrib` is set).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

HFileInfo

Defines a structure which holds catalog information about a file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct HFileInfo {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioACUser;
    FInfo ioFlFndrInfo;
    long ioDirID;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
    unsigned long ioFlBkDat;
    FXInfo ioFlXFndrInfo;
    long ioFlParID;
    long ioFlClpSiz;
};
typedef struct HFileInfo HFileInfo;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`

The file reference number of an open file.

`ioFVersNum`

A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`

Reserved.

`ioFDirIndex`

A file and directory index. If this field contains a positive number, `PBGetCatInfoSync` and `PBGetCatInfoAsync` return information about the file or directory having that directory index in the directory specified by the `ioVRefNum` field. (If `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

If this field contains 0, `PBGetCatInfoSync` or `PBGetCatInfoAsync` returns information about the file or directory whose name is specified in the `ioNamePtr` field and that is located in the directory specified by the `ioVRefNum` field. (Once again, if `ioVRefNum` contains a volume reference number, the specified directory is that volume's root directory.)

If this field contains a negative number, `PBGetCatInfoSync` or `PBGetCatInfoAsync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDirID` field. If both `ioDirID` and `ioVRefNum` are set to 0, `PBGetCatInfoSync` or `PBGetCatInfoAsync` returns information about the current default directory.

`ioFlAttrib`

File or directory attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field.

`ioACUser`

The user's access rights for the specified directory. See "User Privileges Constants" (page 316) for the meaning of the bits in this field.

`ioFlFndrInfo`

Finder information.

`ioDirID`

A directory ID or file ID. On input to `PBGetCatInfoSync` or `PBGetCatInfoAsync`, this field contains a directory ID (which is used only if the `ioFDirIndex` field is negative). On output, this field contains the file ID of the specified file.

`ioFlStBlk`

The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`

The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`

The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`

The first allocation block of the resource fork.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioFlRLgLen`
: The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
: The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
: The date and time of the file's creation, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the creation date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970, and that is the value which will be returned if you later try to retrieve the creation date.

  Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates.

`ioFlMdDat`
: The date and time of the last modification to the file, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the modification date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

`ioFlBkDat`
: The date and time that the file was last backed up, in seconds since midnight, January 1, 1904. However, on Mac OS X, if you set the backup date to a date between January 1, 1904 and January 1, 1970, it will be clipped to January 1, 1970.

  Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates.

`ioFlXFndrInfo`
: Additional Finder information.

`ioFlParID`
: The directory ID of the file's parent directory.

`ioFlClpSiz`
: The clump size to be used when writing the file if it's 0, the volume's clump size is used when the file is opened.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HFileParam

Defines a parameter block used by low-level HFS functions for file creation, deletion, and information retrieval.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct HFileParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioFRefNum;
    SInt8 ioFVersNum;
    SInt8 filler1;
    short ioFDirIndex;
    SInt8 ioFlAttrib;
    SInt8 ioFlVersNum;
    FInfo ioFlFndrInfo;
    long ioDirID;
    unsigned short ioFlStBlk;
    long ioFlLgLen;
    long ioFlPyLen;
    unsigned short ioFlRStBlk;
    long ioFlRLgLen;
    long ioFlRPyLen;
    unsigned long ioFlCrDat;
    unsigned long ioFlMdDat;
};
typedef struct HFileParam HFileParam;
typedef HFileParam * HFileParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioFRefNum`
The file reference number of an open file.

`ioFVersNum`
A file version number. This field is no longer used. File version numbers are an artifact of the obsolete MFS, and are not supported on HFS volumes. You should always set this field to 0.

`filler1`
Reserved.

`ioFDirIndex`
A directory index for use with the `PBHGetFInfoSync` (page 444) and `PBHGetFInfoAsync` (page 442) functions.

`ioFlAttrib`
File attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field.

`ioFlVersNum`
A file version number. This feature is no longer supported, and you must always set this field to 0.

`ioFlFndrInfo`
Information used by the Finder.

`ioDirID`
A directory ID.

`ioFlStBlk`
The first allocation block of the data fork. This field contains 0 if the file's data fork is empty.

`ioFlLgLen`
The logical length (logical end-of-file) of the data fork.

`ioFlPyLen`
The physical length (physical end-of-file) of the data fork.

`ioFlRStBlk`
The first allocation block of the resource fork. This field contains 0 if the file's resource fork is empty.

`ioFlRLgLen`
The logical length (logical end-of-file) of the resource fork.

`ioFlRPyLen`
The physical length (physical end-of-file) of the resource fork.

`ioFlCrDat`
The date and time of the file's creation, specified in seconds since midnight, January 1, 1904.

`ioFlMdDat`
The date and time of the last modification to the file, specified in seconds since midnight, January 1, 1904.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

HFSUniStr255

Used by the File Manager to return Unicode strings.

```
struct HFSUniStr255 {
    UInt16 length;
    UniChar unicode[255];
};
typedef struct HFSUniStr255 HFSUniStr255;
```

Fields

`length`

The number of unicode characters in the string.

`unicode`

The string, in unicode characters.

Discussion

This data type is a string of up to 255 16-bit Unicode characters, with a preceding 16-bit length (number of characters). Note that only the first length characters have meaningful values; the remaining characters may be set to arbitrary values. A caller should always assume that the entire structure will be modified, even if the actual string length is less than 255.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

HIOParam

Defines a parameter block used by low-level HFS I/O functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct HIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    long ioPosOffset;
};
typedef struct HIOParam HIOParam;
typedef HIOParam * HIOParamPtr;
```

Fields

`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
: The file reference number of an open file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVersNum`
A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
The access mode. See "File Access Permission Constants" (page 295).

`ioMisc`
Depending on the function called, this field contains either a logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
A pointer to a data buffer into which data is written by `PBReadSync` and `PBReadAsync` calls, and from which data is read by `PBWriteSync` and `PBWriteAsync` calls.

`ioReqCount`
The requested number of bytes to be read, written, or allocated.

`ioActCount`
The number of bytes actually read, written, or allocated.

`ioPosMode`
The positioning mode (base location) for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 314) to set or test their value.

You can also use the constants described in "Cache Constants" (page 275) to indicate whether or not to cache the data.

`ioPosOffset`
The offset to be used in conjunction with the base location specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

HParamBlockRec

Describes the HFS parameter block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
union HParamBlockRec {
    HIOParam ioParam;
    HFileParam fileParam;
    HVolumeParam volumeParam;
    AccessParam accessParam;
    ObjParam objParam;
    CopyParam copyParam;
    WDParam wdParam;
    FIDParam fidParam;
    CSParam csParam;
    ForeignPrivParam foreignPrivParam;
};
typedef union HParamBlockRec HParamBlockRec;
typedef HParamBlockRec * HParmBlkPtr;
```

Fields

`ioParam`
: A parameter block used by low-level HFS I/O functions. See `HIOParam` (page 241).

`fileParam`
: A parameter block used by low-level HFS functions for file creation, deletion, and information retrieval. See `HFileParam` (page 238).

`volumeParam`
: A parameter block used by low-level HFS volume manipulation functions. See `HVolumeParam` (page 245).

`accessParam`
: A parameter block used by low-level HFS file and directory access rights manipulation functions. See `AccessParam` (page 180).

`objParam`
: A parameter block used by low-level HFS user and group information functions. See `ObjParam` (page 251).

`copyParam`
: A parameter block used by low-level HFS file copying functions. See `CopyParam` (page 191).

`wdParam`
: A parameter block used by low-level HFS working directory functions. See `WDParam` (page 262).

`fidParam`
: A parameter block used by low-level HFS file ID functions. See `FIDParam` (page 204).

`csParam`
: A parameter block used by low-level HFS catalog search functions. See `CSParam` (page 193).

`foreignPrivParam`
: A parameter block used by low-level HFS foreign privileges functions. See `ForeignPrivParam` (page 208).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

HVolumeParam

Defines a parameter block used by low-level HFS volume manipulation functions.

```
struct HVolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler2;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsMod;
    short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVBitMap;
    unsigned short ioAllocPtr;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtCNID;
    unsigned short ioVFrBlk;
    unsigned short ioVSigWord;
    short ioVDrvInfo;
    short ioVDRefNum;
    short ioVFSID;
    unsigned long ioVBkUp;
    short ioVSeqNum;
    unsigned long ioVWrCnt;
    unsigned long ioVFilCnt;
    unsigned long ioVDirCnt;
    long ioVFndrInfo[8];
};
typedef struct HVolumeParam HVolumeParam;
typedef HVolumeParam * HVolumeParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`filler2`

Reserved.

`ioVolIndex`

A volume index for use with the `PBHGetVInfoSync` (page 450) and `PBHGetVInfoAsync` (page 447) functions.

`ioVCrDate`

The date and time of the volume's initialization.

`ioVLsMod`

The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`

The volume attributes. See "Volume Information Attribute Constants" (page 323) for the meanings of the bits in this field.

`ioVNmFls`

The number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.

To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoAsync` (page 423) or `PBGetCatInfoSync` (page 426) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`

The first block of the volume bitmap.

`ioAllocPtr`

The block at which the next new file starts. Used internally.

`ioVNmAlBlks`

The number of allocation blocks.

`ioVAlBlkSiz`

The size of allocation blocks.

`ioVClpSiz`

The clump size.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioAlBlSt`
The first block in the volume map.

`ioVNxtCNID`
The next unused catalog node ID.

`ioVFrBlk`
The number of unused allocation blocks.

`ioVSigWord`
A signature word identifying the type of volume it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.

`ioVDrvInfo`
The drive number of the drive containing the volume.

`ioVDRefNum`
For online volumes, the reference number of the I/O driver for the drive identified by the `ioVDrvInfo` field.

`ioVFSID`
The file-system identifier. It indicates which file system is servicing the volume it's zero for File Manager volumes and nonzero for volumes handled by an external file system.

`ioVBkUp`
The date and time the volume was last backed up; this is 0 if the volume has never been backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
The volume write count.

`ioVFilCnt`
The total number of files on the volume.

`ioVDirCnt`
The total number of directories (not including the root directory) on the volume.

`ioVFndrInfo`
Information used by the Finder.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

IOCompletionUPP

A universal procedure pointer to an application-defined completion function.

```
typedef IOCompletionProcPtr IOCompletionUPP;
```

Discussion
See `IOCompletionProcPtr` (page 179).

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

IOParam

Defines a parameter block used by low-level I/O functions.

```
struct IOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    long ioPosOffset;
};
typedef struct IOParam IOParam;
typedef IOParam * IOParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`
: The file reference number of an open file.

`ioVersNum`
: A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
: The access mode. See "File Access Permission Constants" (page 295).

`ioMisc`
: Depending on the function called, this field contains either a new logical end-of-file (for the `PBGetEOFSync/ PBGetEOFAsync` and `PBSetEOFSync/ PBSetEOFAsync` functions), a new version number, or a pointer to a new pathname (for the `PBHRenameSync/ PBHRenameAsync` functions). Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
: A pointer to a data buffer into which data is written by `PBReadSync` and `PBReadAsync` calls; and from which data is read by `PBWriteSync` and `PBWriteAsync` calls.

`ioReqCount`
: The requested number of bytes to be read, written, or allocated.

`ioActCount`
: The number of bytes actually read, written, or allocated.

`ioPosMode`
: The positioning mode (base location) for positioning the file mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 314) to set or test their value.

  You can also use the constants described in "Cache Constants" (page 275) to indicate whether the data should be cached.

`ioPosOffset`
: The offset to be used in conjunction with the base location specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Files.h`

MultiDevParam

Defines a parameter block used by low-level functions in the classic Device Manager to access multiple devices.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct MultiDevParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioMRefNum;
    SInt8 ioMVersNum;
    SInt8 ioMPermssn;
    Ptr ioMMix;
    short ioMFlags;
    Ptr ioSEBlkPtr;
};
typedef struct MultiDevParam MultiDevParam;
typedef MultiDevParam * MultiDevParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`ioMRefNum`

The driver reference number.

`ioMVersNum`

The slot version number.

`ioMPermssn`

Permissions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioMMix`
Reserved.

`ioMFlags`
Flags specifying the number of additional fields. You should set the `fMulti` bit (bit 0) of this field and clear all of the other bits.

`ioSEBlkPtr`
A pointer to an external parameter block that is customized for the devices installed in the slot.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

ObjParam

Defines a parameter block used by low-level HFS user and group information functions.

```
struct ObjParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler7;
    short ioObjType;
    StringPtr ioObjNamePtr;
    long ioObjID;
};
typedef struct ObjParam ObjParam;
typedef ObjParam * ObjParamPtr;
```

Fields
`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`filler7`

Reserved.

`ioObjType`

A function code. The values passed in this field are determined by the function to which you pass this parameter block.

`ioObjNamePtr`

A pointer to the returned creator/group name.

`ioObjID`

The creator/group ID.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Files.h`

ParamBlockRec

Describes the basic File Manager parameter block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
union ParamBlockRec {
    IOParam ioParam;
    FileParam fileParam;
    VolumeParam volumeParam;
    CntrlParam cntrlParam;
    SlotDevParam slotDevParam;
    MultiDevParam multiDevParam;
};
typedef union ParamBlockRec ParamBlockRec;
typedef ParamBlockRec * ParmBlkPtr;
```

Fields

`ioParam`

`fileParam`

`volumeParam`

`cntrlParam`

`slotDevParam`

`multiDevParam`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

SlotDevParam

Defines a parameter block used by low-level functions in the classic Device Manager to access a single slot device.

```
struct SlotDevParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioSRefNum;
    SInt8 ioSVersNum;
    SInt8 ioSPermssn;
    Ptr ioSMix;
    short ioSFlags;
    SInt8 ioSlot;
    SInt8 ioID;
};
typedef struct SlotDevParam SlotDevParam;
typedef SlotDevParam * SlotDevParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`ioSRefNum`
The driver reference number.

`ioSVersNum`
The slot version number.

`ioSPermssn`
Permissions.

`ioSMix`
Reserved.

`ioSFlags`
Flags determining the number of additional fields. You should clear all of the bits in this field.

`ioSlot`
The slot number.

`ioID`
The slot resource ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VCB

Defines a volume control block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct VCB {
    QElemPtr qLink;
    short qType;
    short vcbFlags;
    unsigned short vcbSigWord;
    unsigned long vcbCrDate;
    unsigned long vcbLsMod;
    short vcbAtrb;
    unsigned short vcbNmFls;
    short vcbVBMSt;
    short vcbAllocPtr;
    unsigned short vcbNmAlBlks;
    long vcbAlBlkSiz;
    long vcbClpSiz;
    short vcbAlBlSt;
    long vcbNxtCNID;
    unsigned short vcbFreeBks;
    Str27 vcbVN;
    short vcbDrvNum;
    short vcbDRefNum;
    short vcbFSID;
    short vcbVRefNum;
    Ptr vcbMAdr;
    Ptr vcbBufAdr;
    short vcbMLen;
    short vcbDirIndex;
    short vcbDirBlk;
    unsigned long vcbVolBkUp;
    unsigned short vcbVSeqNum;
    long vcbWrCnt;
    long vcbXTClpSiz;
    long vcbCTClpSiz;
    unsigned short vcbNmRtDirs;
    long vcbFilCnt;
    long vcbDirCnt;
    long vcbFndrInfo[8];
    unsigned short vcbVCSize;
    unsigned short vcbVBMCSiz;
    unsigned short vcbCtlCSiz;
    unsigned short vcbXTAlBlks;
    unsigned short vcbCTAlBlks;
    short vcbXTRef;
    short vcbCTRef;
    Ptr vcbCtlBuf;
    long vcbDirIDM;
    short vcbOffsM;
};
typedef struct VCB VCB;
typedef VCB * VCBPtr;
```

Fields

`qLink`

A pointer to the next entry in the VCB queue.

`qType`

The queue type. When the volume is mounted and the VCB is created, this field is cleared. Thereafter, bit 7 of this field is set whenever a file on that volume is opened.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`vcbFlags`
: Volume flags. Bit 15 is set if the volume information has been changed by a File Manager call since the volume was last flushed by a `FlushVol` (page 502) call. See "Volume Control Block Flags" (page 321).

`vcbSigWord`
: The volume signature.

`vcbCrDate`
: The date and time of the volume's creation (initialization).

`vcbLsMod`
: The date and time of the volume's last modification. This is not necessarily when the volume was last flushed.

`vcbAtrb`
: The volume attributes.

`vcbNmFls`
: The number of files in the root directory of the volume.

`vcbVBMSt`
: The first block of the volume bitmap.

`vcbAllocPtr`
: The start block of the next allocation search. This field is used internally.

`vcbNmAlBlks`
: The number of allocation blocks in the volume.

`vcbAlBlkSiz`
: The allocation block size, in bytes. This value must always be a multiple of 512 bytes.

`vcbClpSiz`
: The default clump size.

`vcbAlBlSt`
: The first allocation block in the volume.

`vcbNxtCNID`
: The next unused catalog node ID (directory or file ID).

`vcbFreeBks`
: The number of unused allocation blocks on the volume.

`vcbVN`
: The volume name. Note that a volume name can occupy at most 27 characters; this is an exception to the normal file and directory name limit of 31 characters.

`vcbDrvNum`
: The drive number of the drive on which the volume is located. When a mounted drive is placed offline or ejected, this field is set to 0.

`vcbDRefNum`
: The driver reference number of the driver used to access the volume When a volume is ejected, this field is set to the previous value of the `vcbDrvNum` field (and hence is a positive number). When a volume is placed offline, this field is set to the negative of the previous value of the `vcbDrvNum` field (and hence is a negative number).

`vcbFSID`
: An identifier for the file system handling the volume it's zero for volumes handled by the File Manager and nonzero for volumes handled by other file systems.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`vcbVRefNum`
: The volume reference number of the volume.

`vcbMAdr`
: Used internally.

`vcbBufAdr`
: Used internally.

`vcbMLen`
: Used internally.

`vcbDirIndex`
: Used internally.

`vcbDirBlk`
: Used internally.

`vcbVolBkUp`
: The date and time that the volume was last backed up.

`vcbVSeqNum`
: Used internally.

`vcbWrCnt`
: The volume write count.

`vcbXTClpSiz`
: The clump size of the extents overflow file.

`vcbCTClpSiz`
: The clump size of the catalog file.

`vcbNmRtDirs`
: The number of directories in the root directory.

`vcbFilCnt`
: The total number of files on the volume.

`vcbDirCnt`
: The total number of directories on the volume.

`vcbFndrInfo`
: Finder information.

`vcbVCSize`
: Used internally.

`vcbVBMCSiz`
: Used internally.

`vcbCtlCSiz`
: Used internally.

`vcbXTAlBlks`
: The size, in allocation blocks, of the extents overflow file.

`vcbCTAlBlks`
: The size, in allocation blocks, of the catalog file.

`vcbXTRef`
: The path reference number for the extents overflow file.

`vcbCTRef`
: The path reference number for the catalog file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`vcbCtlBuf`
: A pointer to the extents and catalog caches.

`vcbDirIDM`
: The directory last searched.

`vcbOffsM`
: The offspring index at the last search.

Discussion
The volume control block queue is a standard operating system queue that's maintained in the system heap. It contains a volume control block for each mounted volume. A volume control block is a nonrelocatable block that contains volume-specific information.

Each time a volume is mounted, the File Manager reads its volume information from the master directory block and uses the information to build a new volume control block (VCB) in the volume control block queue (unless an ejected or offline volume is being remounted). The File Manager also creates a volume buffer in the system heap. When a volume is placed offline, its buffer is released. When a volume is unmounted, its VCB is removed from the VCB queue as well.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VolMountInfoHeader

Defines a volume mounting information header structure for remote volumes.

```
struct VolMountInfoHeader {
    short length;
    VolumeType media;
};
typedef struct VolMountInfoHeader VolMountInfoHeader;
typedef VolMountInfoHeader * VolMountInfoPtr;
```

Fields
`length`
: The length of the `VolMountInfoHeader` structure, which is the total length of the structure header described here, plus the variable-length location data which follows the header.

`media`
: The volume type of the remote volume. The `AppleShareMediaType` represents an AppleShare volume.

  If you are adding support for the programmatic mounting functions to a non-Macintosh file system, you should register a four-character identifier for your volumes with DTS.

Discussion
To mount a remote server, fill out an `VolMountInfoHeader` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

VolumeMountInfoHeader

Defines an extended volume mounting information header structure for remote volumes.

```
struct VolumeMountInfoHeader {
    short length;
    VolumeType media;
    short flags;
};
typedef struct VolumeMountInfoHeader VolumeMountInfoHeader;
typedef VolumeMountInfoHeader * VolumeMountInfoHeaderPtr;
```

Fields

`length`

The length of the `VolumeMountInfoHeader` structure, which is the total length of the structure header described here, plus the variable-length location data which follows the header.

`media`

The volume type of the remote volume. The `AppleShareMediaType` represents an AppleShare volume.

If you are adding support for the programmatic mounting functions to a non-Macintosh file system, you should register a four-character identifier for your volumes with DTS.

`flags`

The volume mount flags. See "Volume Mount Flags" (page 328).

Discussion
This volume mount info record supersedes the `VolMountInfoHeader` (page 258) structure; `VolMountInfoHeader` is included for compatibility. The `VolumeMountInfoHeader` record allows access to the volume mount flags by foreign filesystem writers.

To mount a remote server, fill out an `VolumeMountInfoHeader` structure using the `PBGetVolMountInfo` function and then pass this structure to the `PBVolumeMount` function to mount the volume.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

VolumeParam

Defines a parameter block used by low-level volume manipulation functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct VolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler2;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsBkUp;
    unsigned short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVDirSt;
    short ioVBlLn;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtFNum;
    unsigned short ioVFrBlk;
};
typedef struct VolumeParam VolumeParam;
typedef VolumeParam * VolumeParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`filler2`
Reserved.

`ioVolIndex`
The volume index.

`ioVCrDate`
The date and time of the volume's initialization.

`ioVLsBkUp`
The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`
The volume attributes. See "Volume Information Attribute Constants" (page 323) for the meanings of the bits in this field.

`ioVNmFls`
The number of files in the root directory.

`ioVDirSt`
The first block of the volume directory.

`ioVBlLn`
Length of directory in blocks.

`ioVNmAlBlks`
The number of allocation blocks.

`ioVAlBlkSiz`
The size of allocation blocks.

`ioVClpSiz`
The volume clump size.

`ioAlBlSt`
The first block in the volume map.

`ioVNxtFNum`
The next unused file number.

`ioVFrBlk`
The number of unused allocation blocks.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

VolumeType

Defines the "signature" of the file system.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
typedef OSType VolumeType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

WDParam

Defines a parameter block used by low-level HFS working directory functions.

```
struct WDParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioWDCreated;
    short ioWDIndex;
    long ioWDProcID;
    short ioWDVRefNum;
    short filler10;
    long filler11;
    long filler12;
    long filler13;
    long ioWDDirID;
};
typedef struct WDParam WDParam;
typedef WDParam * WDParamPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioWDCreated`

`ioWDIndex`
: An index to working directories.

`ioWDProcID`

`ioWDVRefNum`
: The volume reference number for the working directory.

`filler10`
: Reserved.

`filler11`
: Reserved.

`filler12`
: Reserved.

`filler13`
: The working directory's directory ID.

`ioWDDirID`
: The working directory's directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

WDPBRec

Defines a working directory parameter block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct WDPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short filler1;
    short ioWDIndex;
    long ioWDProcID;
    short ioWDVRefNum;
    short filler2[7];
    long ioWDDirID;
};
typedef struct WDPBRec WDPBRec;
typedef WDPBRec * WDPBPtr;
```

Fields

`qLink`
A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
The queue type. This field is used internally by the File Manager.

`ioTrap`
The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`
A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
A volume reference number, 0 for the default volume, or a drive number.

`filler1`
Reserved.

`ioWDIndex`
An index.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioWDProcID`
: An identifier that's used to distinguish between working directories set up by different users you should set `ioWDProcID` to your application's signature.

`ioWDVRefNum`
: The working directory's volume reference number.

`filler2`
: Reserved.

`ioWDDirID`
: The working directory's directory ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XCInfoPBRec

Defines an extended catalog information parameter block.

```
struct XCInfoPBRec {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    ProcPtr ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    long filler1;
    StringPtr ioShortNamePtr;
    short filler2;
    short ioPDType;
    long ioPDAuxType;
    long filler3[2];
    long ioDirID;
};
typedef struct XCInfoPBRec XCInfoPBRec;
typedef XCInfoPBRec * XCInfoPBPtr;
```

Fields
`qLink`
: A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`
: The queue type. This field is used internally by the File Manager.

`ioTrap`
: The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`
: The address of the function that was called. This field is used internally by the File Manager.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`filler1`
: Reserved; set this field to 0.

`ioShortNamePtr`
: A pointer to a Pascal string buffer, of a minimum 13 bytes, which holds the file or directory's short name (MS-DOS format name). This field is required and cannot be `NULL`.

`filler2`
: Reserved; set this field to 0.

`ioPDType`
: The ProDOS file type of the file or directory.

`ioPDAuxType`
: The ProDOS auxiliary type of the file or directory.

`filler3`
: Reserved; set this field to 0.

`ioDirID`
: A directory ID.

Discussion
The `PBGetXCatInfoSync` and `PBGetXCatInfoAsync` functions use this parameter block to return the short name and ProDOS information for files and directories.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XIOParam

Defines an extended I/O parameter block structure.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct XIOParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    short ioRefNum;
    SInt8 ioVersNum;
    SInt8 ioPermssn;
    Ptr ioMisc;
    Ptr ioBuffer;
    long ioReqCount;
    long ioActCount;
    short ioPosMode;
    wide ioWPosOffset;
};
typedef struct XIOParam XIOParam;
typedef XIOParam * XIOParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

`ioResult`

The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`

A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`

A volume reference number, 0 for the default volume, or a drive number.

`ioRefNum`

The file reference number of an open file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVersNum`
: A version number. This field is no longer used and you should always set it to 0.

`ioPermssn`
: The access mode. See "File Access Permission Constants" (page 295).

`ioMisc`
: Depending on the function called, this field contains either a logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer`
: A pointer to a data buffer into which data is written by `_Read` calls and from which data is read by `_Write` calls.

`ioReqCount`
: The requested number of bytes to be read or written.

`ioActCount`
: The number of bytes actually read or written.

`ioPosMode`
: The positioning mode (base location) for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the constants described in "Position Mode Constants" (page 314) to set or test their value. For the functions which use this parameter block, you must have the `kUseWidePositioning` bit set. See "Large Volume Constants" (page 312) for a description of this and other constants.

  You can also use the constants described in "Cache Constants" (page 275) to indicate whether or not to cache the data.

`ioWPosOffset`
: The wide positioning offset to be used in conjunction with the positioning mode specified in the `ioPosMode` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

XVolumeParam

Defines an extended volume information parameter block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
struct XVolumeParam {
    QElemPtr qLink;
    short qType;
    short ioTrap;
    Ptr ioCmdAddr;
    IOCompletionUPP ioCompletion;
    volatile OSErr ioResult;
    StringPtr ioNamePtr;
    short ioVRefNum;
    unsigned long ioXVersion;
    short ioVolIndex;
    unsigned long ioVCrDate;
    unsigned long ioVLsMod;
    short ioVAtrb;
    unsigned short ioVNmFls;
    unsigned short ioVBitMap;
    unsigned short ioAllocPtr;
    unsigned short ioVNmAlBlks;
    unsigned long ioVAlBlkSiz;
    unsigned long ioVClpSiz;
    unsigned short ioAlBlSt;
    unsigned long ioVNxtCNID;
    unsigned short ioVFrBlk;
    unsigned short ioVSigWord;
    short ioVDrvInfo;
    short ioVDRefNum;
    short ioVFSID;
    unsigned long ioVBkUp;
    short ioVSeqNum;
    unsigned long ioVWrCnt;
    unsigned long ioVFilCnt;
    unsigned long ioVDirCnt;
    long ioVFndrInfo[8];
    UInt64 ioVTotalBytes;
    UInt64 ioVFreeBytes;
};
typedef struct XVolumeParam XVolumeParam;
typedef XVolumeParam * XVolumeParamPtr;
```

Fields

`qLink`

A pointer to the next entry in the file I/O queue. (This field is used internally by the File Manager to keep track of asynchronous calls awaiting execution.

`qType`

The queue type. This field is used internally by the File Manager.

`ioTrap`

The trap number of the function that was called. This field is used internally by the File Manager.

`ioCmdAddr`

The address of the function that was called. This field is used internally by the File Manager.

`ioCompletion`

A universal procedure pointer to a completion routine to be executed at the end of an asynchronous call. It should be 0 for asynchronous calls with no completion routine and is automatically set to 0 for all synchronous calls. See `IOCompletionProcPtr` (page 179) for information about completion routines.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
: The result code of the function. For synchronous calls, this field is the same as the result code of the function call itself. To determine when an asynchronous call has actually been completed, your application can poll this field it's set to a positive number when the call is made and receives the actual result code when the call is completed.

`ioNamePtr`
: A pointer to a pathname. Whenever a function description specifies that `ioNamePtr` is used—whether for input, output, or both—it's very important that you set this field to point to storage for a `Str255` value (if you're using a pathname) or to `NULL` (if you're not).

`ioVRefNum`
: A volume reference number, 0 for the default volume, or a drive number.

`ioXVersion`
: The version of the `XVolumeParam` parameter block; currently, this is 0.

`ioVolIndex`
: A volume index for use with the `PBXGetVolInfoSync` (page 497) and `PBXGetVolInfoAsync` (page 494) functions.

`ioVCrDate`
: The date and time that the volume was created (initialized).

`ioVLsMod`
: The date and time that the volume information was last modified. This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.

`ioVAtrb`
: The volume attributes. See "Volume Information Attribute Constants" (page 323) for the meanings of the bits in this field.

`ioVNmFls`
: The number of files in the root directory.

`ioVBitMap`
: The first block of the volume bitmap.

`ioAllocPtr`
: The block at which the next new file starts. Used internally.

`ioVNmAlBlks`
: The number of allocation blocks.

`ioVAlBlkSiz`
: The size of the allocation blocks.

`ioVClpSiz`
: The clump size.

`ioAlBlSt`
: The first block in the volume map.

`ioVNxtCNID`
: The next unused catalog node ID.

`ioVFrBlk`
: The number of unused allocation blocks.

`ioVSigWord`
: A signature word identifying the type of volume it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVDrvInfo`
The drive number of the drive containing the volume.

`ioVDRefNum`
For online volumes, the reference number of the I/O driver for the drive identified by the `ioVDrvInfo` field.

`ioVFSID`
The file-system identifier. It indicates which file system is servicing the volume it's zero for File Manager volumes and nonzero for volumes handled by an external file system.

`ioVBkUp`
The date and time that the volume was last backed up; this is 0 if the volume has never been backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
The volume write count.

`ioVFilCnt`
The total number of files on the volume.

`ioVDirCnt`
The total number of directories (not including the root directory) on the volume.

`ioVFndrInfo`
Information used by the Finder.

`ioVTotalBytes`
The total number of bytes on the volume.

`ioVFreeBytes`
The number of free bytes on the volume.

Discussion
The functions `PBXGetVolInfoSync` and `PBXGetVolInfoAsync` use this parameter block structure to pass arguments and return values.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Files.h`

Constants

AFP Tag Length Constants

Specify the length of tagged address information for AppleShare volumes.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kAFPTagLengthIP = 0x06,
    kAFPTagLengthIPPort = 0x08,
    kAFPTagLengthDDP = 0x06
};
```

Constants

`kAFPTagLengthIP`
The length of a 4 byte IP address.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagLengthIPPort`
The length of a 4 byte IP address and a 2 byte port.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagLengthDDP`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion
These constants are used in the `fLength` field of the `AFPTagData` (page 182) structure to indicate the length, in bytes, of the tagged address information. This length includes the fLength field itself.

AFP Tag Type Constants

Specify the type of tagged address information for AppleShare clients.

```
enum {
    kAFPTagTypeIP = 0x01,
    kAFPTagTypeIPPort = 0x02,
    kAFPTagTypeDDP = 0x03,
    kAFPTagTypeDNS = 0x04
};
```

Constants

`kAFPTagTypeIP`
A basic 4 byte IP address, most significant byte first.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagTypeIPPort`
A 4 byte IP address and a 2 byte port number, most significant byte first.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kAFPTagTypeDDP`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kAFPTagTypeDNS`
The address is a DNS name in address:port format. The total length of the DNS name is variable up to 254 characters.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `fType` field of the tagged address structure, `AFPTagData` (page 182), to specify the type of address represented by the structure.

Allocation Flags

Indicate how new space is to be allocated.

```
typedef UInt16 FSAllocationFlags;
enum {
    kFSAllocDefaultFlags = 0x0000,
    kFSAllocAllOrNothingMask = 0x0001,
    kFSAllocContiguousMask = 0x0002,
    kFSAllocNoRoundUpMask = 0x0004,
    kFSAllocReservedMask = 0xFFF8
};
```

Constants
`kFSAllocDefaultFlags`
Allocate as much as possible, not necessarily contiguous.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocAllOrNothingMask`
This bit is set when an allocation must allocate the total requested amount, or else fail with nothing allocated; when this bit is not set, the allocation may complete successfully but allocate less than requested.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocContiguousMask`
This bit is set when an allocation should allocate one contiguous range of space on the volume. If this bit is clear, multiple discontiguous extents may be allocated to fulfill the request.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSAllocNoRoundUpMask`
This bit is set when an allocation should no round up to the clump size. If this bit is clear, then additional space beyond the amount requested may be allocated; this is done by some volume formats (including HFS and HFS Plus) to avoid many small allocation requests. If the bit is set, no additional allocation is done (except where required by the volume format, such as rounding up to a multiple of the allocation block size).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSAllocReservedMask`
Reserved; set to zero.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
If the `kFSAllocContiguousMask` bit is set, then then any newly allocated space must be in one contiguous extent (preferably contiguous with any space already allocated). If `kFSAllocAllOrNothingMask` is set, then the entire `requestCount` bytes must be allocated for the call to succeed; if not set, as many bytes as possible will be allocated (without error). If `kFSAllocNoRoundUpMask` is set, then no additional space is allocated (such as rounding up to a multiple of a clump size); if clear, the volume format may allocate more space than requested as an attempt to reduce fragmentation.

AppleShare Volume Signature

Defines the volume signature for AppleShare volumes.

```
enum {
    AppleShareMediaType = 'afpm'
};
```

Authentication Method Constants

Define the login methods for remote volumes.

```
enum {
    kNoUserAuthentication = 1,
    kPassword = 2,
    kEncryptPassword = 3,
    kTwoWayEncryptPassword = 6
};
```

Constants
`kNoUserAuthentication`
No password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kPassword`
8-byte password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kEncryptPassword`
Encrypted 8-byte password.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kTwoWayEncryptPassword`
Two-way random encryption.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
These constants are used in the `uamType` field of an `AFPVolMountInfo` (page 183) structure and in the `ioObjType` field of the parameter block passed to the `PBHGetLogInInfoSync` and `PBHGetLogInInfoAsync` functions to specify the type of user authentication used by a remote volume.

Cache Constants

Indicate whether or not data should be cached.

```
enum {
    pleaseCacheBit = 4,
    pleaseCacheMask = 0x0010,
    noCacheBit = 5,
    noCacheMask = 0x0020,
    rdVerifyBit = 6,
    rdVerifyMask = 0x0040,
    rdVerify = 64,
    forceReadBit = 6,
    forceReadMask = 0x0040,
    newLineBit = 7,
    newLineMask = 0x0080,
    newLineCharMask = 0xFF00
};
```

Constants

`pleaseCacheBit`
Indicates that the data should be cached.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`pleaseCacheMask`
Requests that the data be cached, if possible. You should cache reads and writes if you read or write the same portion of a file multiple times.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`noCacheBit`
Indicates that data should not be cached.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`noCacheMask`
Requests that the data not be cached, if possible. You should not cache reads and writes if you read or write data from a file only once.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`rdVerifyBit`
Indicates that all reads should come from the source and be verified against the data in memory.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`rdVerifyMask`

Requests that all reads (not writes) come directly from the source and be verified against the data in memory. This flushes the cache and sends all read requests to the data source.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`rdVerify`

This is the old name of `rdVerifyMask`. Both request that all reads come directly from the source of the data and be compared against the data in memory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`forceReadBit`

Indicates that reads should come from the disk.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`forceReadMask`

Forces reads from disk, bypassing all caches. Clients can use this to verify that data is stored correctly on the media (for example, to verify after writing) by reading the data into a different buffer while setting the bit, and then comparing the newly read data with the previously written data.

The `forceReadMask` is the same as the `rdVerifyMask` used in the older APIs. The actual implementation of the `rdVerifyMask` in the older APIs actually caused the "force read" behavior, and only compared the data in partial sectors. `FSReadFork` cleans up this behavior by always letting the client do all of the compares.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineBit`

Indicates that newline mode should be used for reads.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineMask`

Requests that newline mode be used for reads. In newline mode, the read stops when one of the following conditions is met:

- The requested number of bytes have been read.
- The end-of-file is reached.
- The newline character has been read. If the newline character is found, it will be the last character put into the buffer and the number of bytes read will include it.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`newLineCharMask`

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion

For the `FSReadFork` and `FSWriteFork` functions, and their parameter block equivalents, you may add either of the `pleaseCacheMask` or `noCacheMask` constants to one of the "Position Mode Constants" (page 314) to hint whether the data should be cached or not.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

The `pleaseCacheBit` and the `noCacheBit` are mutually exclusive and only one should be set at a time. If neither bit is set, the program has indicated that it doesn't care if the data is cached or not.

Catalog Information Bitmap Constants

Specify what file or fork information to get or set.

```
enum {
    kFSCatInfoNone = 0x00000000,
    kFSCatInfoTextEncoding = 0x00000001,
    kFSCatInfoNodeFlags = 0x00000002,
    kFSCatInfoVolume = 0x00000004,
    kFSCatInfoParentDirID = 0x00000008,
    kFSCatInfoNodeID = 0x00000010,
    kFSCatInfoCreateDate = 0x00000020,
    kFSCatInfoContentMod = 0x00000040,
    kFSCatInfoAttrMod = 0x00000080,
    kFSCatInfoAccessDate = 0x00000100,
    kFSCatInfoBackupDate = 0x00000200,
    kFSCatInfoPermissions = 0x00000400,
    kFSCatInfoFinderInfo = 0x00000800,
    kFSCatInfoFinderXInfo = 0x00001000,
    kFSCatInfoValence = 0x00002000,
    kFSCatInfoDataSizes = 0x00004000,
    kFSCatInfoRsrcSizes = 0x00008000,
    kFSCatInfoSharingFlags = 0x00010000,
    kFSCatInfoUserPrivs = 0x00020000,
    kFSCatInfoUserAccess = 0x00080000,
    kFSCatInfoSetOwnership = 0x00100000,
    kFSCatInfoAllDates = 0x000003E0,
    kFSCatInfoGettableInfo = 0x0003FFFF,
    kFSCatInfoSettableInfo = 0x00001FE3,
    kFSCatInfoReserved = 0xFFFC0000
};
```

Constants

`kFSCatInfoNone`

No catalog information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoTextEncoding`

Retrieve or set the text encoding hint, in the `textEncodingHint` field.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoNodeFlags`

Retrieve or set the catalog node flags. Currently, you can only set bits 0 and 4. See "Catalog Information Node Flags" (page 280) for more information on these flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSCatInfoVolume`
Retrieve the volume reference number of the volume on which the file or directory resides.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoParentDirID`
Retrieve the parent directory ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoNodeID`
Retrieve the file or directory ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoCreateDate`
Retrieve or set the creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoContentMod`
Retrieve or set the date that the resource or data fork was last modified.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoAttrMod`
Retrieve or set the date that an attribute or named fork was last modified.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoAccessDate`
Retrieve or set the date that the fork or file was last accessed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoBackupDate`
Retrieve or set the date that the fork or file was last backed up.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoPermissions`
Retrieve or set the file or fork's permissions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoFinderInfo`
Retrieve or set the file or fork's Finder information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSCatInfoFinderXInfo`
Retrieve or set the file or fork's extended Finder information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoValence`
For folders only, retrieve the valence of the folder. For files, this is zero.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoDataSizes`
Retrieve the logical and physical size of the data fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoRsrcSizes`
Retrieve the logical and physical size of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoSharingFlags`
Retrieve the fork or file's sharing flags: `kioFlAttribMountedBit`, `kioFlAttribSharePointBit`. See "File Attribute Constants" (page 300) for more information on these bits.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoUserPrivs`
Retrieve the file's user privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoUserAccess`
Available in Mac OS X v10.1 and later.

Declared in `Files.h`.

`kFSCatInfoSetOwnership`
Attempt to set the file's user and group (UID and GID). If the File Manager cannot set the the user or group ID, the call fails. (Mac OS X only).

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`kFSCatInfoAllDates`
Retrieve or set all of the date information for the fork or file: creation date, modification dates, access date, backup date, etc.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoGettableInfo`
Retrieve all gettable data.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSCatInfoSettableInfo`
Set all settable data. This includes the flags, dates, permissions, Finder info, and text encoding hint.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSCatInfoReserved`
Represents bits that are currently reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `FSCatalogInfoBitmap` type to specify what file or fork information to get or set. If used with the `FSGetCatalogInfo` or `FSGetCatalogInfoBulk` functions, these constants tell the File Manager which fields to return information in. If used with the `FSSetCatalogInfo` function, these constants tell the File Manager which fields you've filled out with values that it should use to change the fork or file's catalog information.

Catalog Information Node Flags

Define the values used in the `nodeFlags` field of the `FSCatalogInfo` structure.

```
enum {
    kFSNodeLockedBit = 0,
    kFSNodeLockedMask = 0x0001,
    kFSNodeResOpenBit = 2,
    kFSNodeResOpenMask = 0x0004,
    kFSNodeDataOpenBit = 3,
    kFSNodeDataOpenMask = 0x0008,
    kFSNodeIsDirectoryBit = 4,
    kFSNodeIsDirectoryMask = 0x0010,
    kFSNodeCopyProtectBit = 6,
    kFSNodeCopyProtectMask = 0x0040,
    kFSNodeForkOpenBit = 7,
    kFSNodeForkOpenMask = 0x0080,
    kFSNodeHardLinkBit = 8,
    kFSNodeHardLinkMask = 0x00000100
};
```

Constants
`kFSNodeLockedBit`
Set if the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeLockedMask`
Indicates that the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeResOpenBit`
Set if the resource fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSNodeResOpenMask`
Indicates that the resource fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeDataOpenBit`
Set if the data fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeDataOpenMask`
Indicates that the data fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeIsDirectoryBit`
Set if the object is a directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeIsDirectoryMask`
Indicates that the object is a directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeCopyProtectBit`
Set of the file or directory is copy protected.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeCopyProtectMask`
Indicates that the file or directory is copy protected.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeForkOpenBit`
Set if the file or directory has any open fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeForkOpenMask`
Indicates that the file or directory has an open fork of any type.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSNodeHardLinkBit`
Available in Mac OS X v10.2 and later.

Declared in `Files.h`.

`kFSNodeHardLinkMask`
Available in Mac OS X v10.2 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Catalog Information Sharing Flags

Indicate the status of a shared directory.

```
enum {
    kFSNodeInSharedBit = 2,
    kFSNodeInSharedMask = 0x0004,
    kFSNodeIsMountedBit = 3,
    kFSNodeIsMountedMask = 0x0008,
    kFSNodeIsSharePointBit = 5,
    kFSNodeIsSharePointMask = 0x0020
};
```

Constants

`kFSNodeInSharedBit`
Set if a directory is within a share point.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeInSharedMask`
Indicates that the directory is within a share point.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsMountedBit`
Set if a directory is a share point currently mounted by some user.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsMountedMask`
Indicates that the directory is a share point currently mounted by some user.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsSharePointBit`
Set if a directory is a share point (an exported volume).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSNodeIsSharePointMask`
Indicates that the directory is a share point (an exported volume).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

The `FSCatalogInfo` (page 212) structure uses these constants in its `sharingFlags` field.

Catalog Search Bits

Indicate the criteria for a catalog search.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    fsSBPartialNameBit = 0,
    fsSBFullNameBit = 1,
    fsSBFlAttribBit = 2,
    fsSBFlFndrInfoBit = 3,
    fsSBFlLgLenBit = 5,
    fsSBFlPyLenBit = 6,
    fsSBFlRLgLenBit = 7,
    fsSBFlRPyLenBit = 8,
    fsSBFlCrDatBit = 9,
    fsSBFlMdDatBit = 10,
    fsSBFlBkDatBit = 11,
    fsSBFlXFndrInfoBit = 12,
    fsSBFlParIDBit = 13,
    fsSBNegateBit = 14,
    fsSBDrUsrWdsBit = 3,
    fsSBDrNmFlsBit = 4,
    fsSBDrCrDatBit = 9,
    fsSBDrMdDatBit = 10,
    fsSBDrBkDatBit = 11,
    fsSBDrFndrInfoBit = 12,
    fsSBDrParIDBit = 13
};
```

Constants

`fsSBPartialNameBit`
Indicates a search by a substring of the name.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFullNameBit`
Indicates a search by the full name.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlAttribBit`
Indicates a search by the file or directory attributes.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlFndrInfoBit`
For files only indicates a search by the file's Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlLgLenBit`
For files only; indicates a search by the logical length of the data fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlPyLenBit`
For files only; indicates a search by the physical length of the data fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsSBFlRLgLenBit`
For files only; indicates a search for the logical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlRPyLenBit`
For files only; indicates a search by the physical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlCrDatBit`
For files only indicates a search by the file's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlMdDatBit`
For files only indicates a search by the date of the file's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlBkDatBit`
For files only indicates a search by the date of the file's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlXFndrInfoBit`
For files only indicates a search by the file's extended Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlParIDBit`
For files only indicates a search by the file's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBNegateBit`
Indicates a search for all non-matches. That is, if a file or directory matches one of the other specified criteria, it is not returned; if it does not match any of the specified criteria, it is returned.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrUsrWdsBit`
For directories only indicates a search by the directory's Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrNmFlsBit`
For directories only; indicates a search by the number of files in the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsSBDrCrDatBit`
For directories only indicates a search by the directory's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrMdDatBit`
For directories only indicates a search by the date of the directory's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrBkDatBit`
For directories only indicates a search by the date of the directory's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrFndrInfoBit`
For directories only indicates a search by the directory's additional Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrParIDBit`
For directories only indicates a search by the directory's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Catalog Search Constants

Specify the which catalog information fields to use as search criteria.

```
enum {
    fsSBNodeID = 0x00008000,
    fsSBAttributeModDate = 0x00010000,
    fsSBAccessDate = 0x00020000,
    fsSBPermissions = 0x00040000,
    fsSBNodeIDBit = 15,
    fsSBAttributeModDateBit = 16,
    fsSBAccessDateBit = 17,
    fsSBPermissionsBit = 18
};
```

Constants

`fsSBNodeID`
Search by a range of catalog node ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBAttributeModDate`
Search by a range of attribute (fork) modification date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsSBAccessDate`
Search by a range of access date.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBPermissions`
Search by a value or mask of permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBNodeIDBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBAttributeModDateBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBAccessDateBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBPermissionsBit`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Catalog Search Masks

Specify the criteria for a catalog search.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    fsSBPartialName = 1,
    fsSBFullName = 2,
    fsSBFlAttrib = 4,
    fsSBFlFndrInfo = 8,
    fsSBFlLgLen = 32,
    fsSBFlPyLen = 64,
    fsSBFlRLgLen = 128,
    fsSBFlRPyLen = 256,
    fsSBFlCrDat = 512,
    fsSBFlMdDat = 1024,
    fsSBFlBkDat = 2048,
    fsSBFlXFndrInfo = 4096,
    fsSBFlParID = 8192,
    fsSBNegate = 16384,
    fsSBDrUsrWds = 8,
    fsSBDrNmFls = 16,
    fsSBDrCrDat = 512,
    fsSBDrMdDat = 1024,
    fsSBDrBkDat = 2048,
    fsSBDrFndrInfo = 4096,
    fsSBDrParID = 8192
};
```

Constants

`fsSBPartialName`
Search by a substring of the name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFullName`
Search by the full name.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlAttrib`
Search by the file or directory attributes. You can use the attributes to specify that you are searching for a directory, or for a file or directory that is locked by software.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlFndrInfo`
For files only search by the file's Finder info.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlLgLen`
For files only; search by the logical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`fsSBFlPyLen`
For files only; search by the physical length of the data fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsSBFlRLgLen`
For files only; search for the logical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlRPyLen`
For files only; search by the physical length of the resource fork.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlCrDat`
For files only search by the file's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlMdDat`
For files only search by the date of the file's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlBkDat`
For files only search by the date of the file's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlXFndrInfo`
For files only search by the file's extended Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBFlParID`
For files only search by the file's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBNegate`
Search for all non-matches. That is, if a file or directory matches one of the other specified criteria, it is not returned; if it does not match any of the specified criteria, it is returned.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrUsrWds`
For directories only search by the directory's Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrNmFls`
For directories only; search by the number of files in the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsSBDrCrDat`
For directories only search by the directory's creation date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrMdDat`
For directories only search by the date of the directory's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrBkDat`
For directories only search by the date of the directory's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrFndrInfo`
For directories only search by the directory's additional Finder info.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsSBDrParID`
For directories only search by the directory's parent ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Use these constants in the `ioSearchBits` field of the `PBCatSearchSync` and `PBCatSearchAsync` functions to specify the criteria for your search.

Extended AFP Volume Mounting Information Flag

Specifies a flag used in the `extendedFlags` field of the `AFPXVolMountInfo` structure.

```
enum {
    kAFPExtendedFlagsAlternateAddressMask = 1
};
```

Constants
`kAFPExtendedFlagsAlternateAddressMask`
Indicates that the `alternateAddressOffset` field in the `AFPXVolMountInfo` record is used.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
See the `AFPXVolMountInfo` (page 185) structure for more information.

Extended Volume Attributes

Describe a volume's extended attributes.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    bIsEjectable = 0,
    bSupportsHFSPlusAPIs = 1,
    bSupportsFSCatalogSearch = 2,
    bSupportsFSExchangeObjects = 3,
    bSupports2TBFiles = 4,
    bSupportsLongNames = 5,
    bSupportsMultiScriptNames = 6,
    bSupportsNamedForks = 7,
    bSupportsSubtreeIterators = 8,
    bL2PCanMapFileBlocks = 9,
    bParentModDateChanges = 10,
    bAncestorModDateChanges = 11,
    bSupportsSymbolicLinks = 13,
    bIsAutoMounted = 14,
    bAllowCDiDataHandler = 17,
    bSupportsExclusiveLocks = 18,
    bSupportsJournaling = 19,
    bNoVolumeSizes = 20,
    bIsOnInternalBus = 21,
    bIsCaseSensitive = 22,
    bIsCasePreserving = 23,
    bDoNotDisplay = 24,
    bIsRemovable = 25,
    bNoRootTimes = 26,
    bIsOnExternalBus = 27,
    bSupportsExtendedFileSecurity = 28
};
```

Constants

`bIsEjectable`

The volume is in an ejectable disk drive .

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsHFSPlusAPIs`

The volume supports the HFS Plus APIs directly, i.e., the File Manager does not emulate them.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsFSCatalogSearch`

The volume supports the `FSCatalogSearch` (page 45) operation.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsFSExchangeObjects`

The volume supports the `FSExchangeObjects` (page 59) function.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupports2TBFiles`

The volume supports 2 terabyte files.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bSupportsLongNames`
The volume supports file, directory, and volume names longer than 31 characters.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsMultiScriptNames`
The volume supports file, directory, and volume names with characters from multiple script systems.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsNamedForks`
The volume supports named forks other than the data and resource forks.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsSubtreeIterators`
The volume supports recursive iterators, not at the volume root.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bL2PCanMapFileBlocks`
The volume supports the `Lg2Phys` SPI correctly.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bParentModDateChanges`
On this volume, changing a file or folder causes its parent's modification date to change.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAncestorModDateChanges`
On this volume, changing a file or folder causes all ancestor modification dates to change.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bSupportsSymbolicLinks`
The volume supports the creation and use of symbolic links (Mac OS X only).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bIsAutoMounted`
The volume was mounted automatically (Mac OS X only).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAllowCDiDataHandler`
QuickTime's CDi data handler is allowed to examine the volume.

Available in Mac OS X v10.1 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bSupportsExclusiveLocks`
The volume supports exclusive access to files opened for writing.

Available in Mac OS X v10.2 and later.

Declared in `Files.h`.

`bSupportsJournaling`
The volume supports journaling. This does not indicate whether journaling is currently enabled on the volume.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`bNoVolumeSizes`
The volume is unable to report volume size or free space.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`bIsOnInternalBus`
The device is on an internal bus. See note below.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`bIsCaseSensitive`
The volume is case-sensitive.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`bIsCasePreserving`
The volume is preserves case.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`bDoNotDisplay`
The volume should not be displayed in the user interface.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

`bIsRemovable`
The device is removable according to IOKit.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`bNoRootTimes`
The volume does not set reliable times for its root directory.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`bIsOnExternalBus`
The device is on an external bus. See note below.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bSupportsExtendedFileSecurity`
The volume supports FSFileSecurity objects.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
The `GetVolParmsInfoBuffer` (page 233) structure uses these constants in its `vMExtendedAttributes` field.

Note: A volume can return one of four states via the `bIsOnInternalBus` and `bIsOnExternalBus` bits. A volume known to be on an internal bus will set `bIsOnInternalBus` and clear `bIsOnExternalBus`. A volume known to be on an external bus will clear `bIsOnInternalBus` and set `bIsOnExternalBus`. A volume on a bus that is indeterminate (could be either) will set both bits. A volume not on a local bus, such as a network volume, will leave both bits clear.

FCB Flags

Specify flags that describe the state of a file.

```
enum {
    kioFCBWriteBit = 8,
    kioFCBWriteMask = 0x0100,
    kioFCBResourceBit = 9,
    kioFCBResourceMask = 0x0200,
    kioFCBWriteLockedBit = 10,
    kioFCBWriteLockedMask = 0x0400,
    kioFCBLargeFileBit = 11,
    kioFCBLargeFileMask = 0x0800,
    kioFCBSharedWriteBit = 12,
    kioFCBSharedWriteMask = 0x1000,
    kioFCBFileLockedBit = 13,
    kioFCBFileLockedMask = 0x2000,
    kioFCBOwnClumpBit = 14,
    kioFCBOwnClumpMask = 0x4000,
    kioFCBModifiedBit = 15,
    kioFCBModifiedMask = 0x8000
};
```

Constants
`kioFCBWriteBit`
Set if data can be written to this file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFCBWriteMask`
Tests if data can be written to this file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioFCBResourceBit`
Set if this FCB describes a resource fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBResourceMask`
Tests if this FCB describes a resource fork.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBWriteLockedBit`
Set if this file has a locked byte range.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBWriteLockedMask`
Tests if this file has a locked byte range.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBLargeFileBit`
Set if this file may grow beyond 2GB and the cache uses file blocks, not bytes.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBLargeFileMask`
Tests if this file may grow beyond 2GB and the cache uses file blocks, not bytes.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBSharedWriteBit`
Set if this file has shared write permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBSharedWriteMask`
Tests if this file has shared write permissions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBFileLockedBit`
Set if this file is locked (write-protected).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kioFCBFileLockedMask`
Tests if this file is locked (write-protected).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioFCBOwnClumpBit`
Set if this file's clump size is specified in the FCB.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFCBOwnClumpMask`
Tests if this file's clump size is specified in the FCB.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFCBModifiedBit`
Set if this file has changed since it was last flushed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFCBModifiedMask`
Tests if this file has changed since it was last flushed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioFCBFlags` field of the `FCBPBRec` (page 202) returned by the functions `PBGetFCBInfoSync` and `PBGetFCBInfoAsync`.

File Access Permission Constants

Specify the type of read and write access to a file or fork.

```
enum {
    fsCurPerm = 0x00,
    fsRdPerm = 0x01,
    fsWrPerm = 0x02,
    fsRdWrPerm = 0x03,
    fsRdWrShPerm = 0x04,
    fsRdDenyPerm = 0x10,
    fsWrDenyPerm = 0x20
};
```

Constants
`fsCurPerm`
Requests whatever permissions are currently allowed. If write access in unavailable (because the file is locked or the file is already open with write permission), then read permission is granted. Otherwise read/write permission is granted.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdPerm`
Requests permission to read the file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsWrPerm`

Requests permission to write to the file. If write permission is granted, no other access paths are granted write permission. Note, however, that the File Manager does not support write-only access to a file. Thus, `fsWrPerm` is synonymous with `fsRdWrPerm`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdWrPerm`

Requests exclusive read and write permission. If exclusive read/ write permission is granted, no other users are granted permission to write to the file. Other users may, however, be granted permission to read the file.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdWrShPerm`

Requests shared read and write permission. Shared read and write permission allows multiple access paths for reading and writing. This is safe only if there is some way of locking portions of the file before writing to them. On volumes that support range locking, you can use the functions `PBLockRangeSync` and `PBUnlockRangeSync` to lock and unlock ranges of bytes within a file. Applications running in Mac OS X version 10.4 or later should use the functions `FSLockRange` and `FSUnlockRange` for this purpose.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRdDenyPerm`

Requests that any other paths be prevented from having read access. A path cannot be opened if you request read permission (with the `fsRdPerm` constant) but some other path has requested deny-read access. Similarly, the path cannot be opened if you request deny-read permission, but some other path already has read access. This constant is only supported on volumes which return the `bHasOpenDeny` attribute when you call `FSGetVolumeParms`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsWrDenyPerm`

Requests that any other paths be prevented from having write access. A path cannot be opened if you request write permission (with the `fsWrPerm` constant) but some other path has requested deny-write access. Similarly, the path cannot be opened if you request deny-write permission, but some other path already has write access. This constant is only supported on volumes which return the `bHasOpenDeny` attribute when you call `FSGetVolumeParms`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Use these constants to request a type of access to a file or fork, or to deny a type of access to a file or fork to other paths that may request access.

Note that it is possible, in Mac OS 8 and 9, to open a file residing on read-only media with write access. In Mac OS X, however, you cannot open a file with write access on read-only media; the attempt to open the file fails with a `wrPermErr` error.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

File and Folder Access Privilege Constants

Specify access privileges for files and directories in the `ioACAccess` field of the `AccessParam` data type.

```
enum {
    kioACAccessOwnerBit = 31,
    kioACAccessOwnerMask = 0x80000000,
    kioACAccessBlankAccessBit = 28,
    kioACAccessBlankAccessMask = 0x10000000,
    kioACAccessUserWriteBit = 26,
    kioACAccessUserWriteMask = 0x04000000,
    kioACAccessUserReadBit = 25,
    kioACAccessUserReadMask = 0x02000000,
    kioACAccessUserSearchBit = 24,
    kioACAccessUserSearchMask = 0x01000000,
    kioACAccessEveryoneWriteBit = 18,
    kioACAccessEveryoneWriteMask = 0x00040000,
    kioACAccessEveryoneReadBit = 17,
    kioACAccessEveryoneReadMask = 0x00020000,
    kioACAccessEveryoneSearchBit = 16,
    kioACAccessEveryoneSearchMask = 0x00010000,
    kioACAccessGroupWriteBit = 10,
    kioACAccessGroupWriteMask = 0x00000400,
    kioACAccessGroupReadBit = 9,
    kioACAccessGroupReadMask = 0x00000200,
    kioACAccessGroupSearchBit = 8,
    kioACAccessGroupSearchMask = 0x00000100,
    kioACAccessOwnerWriteBit = 2,
    kioACAccessOwnerWriteMask = 0x00000004,
    kioACAccessOwnerReadBit = 1,
    kioACAccessOwnerReadMask = 0x00000002,
    kioACAccessOwnerSearchBit = 0,
    kioACAccessOwnerSearchMask = 0x00000001,
    kfullPrivileges = 0x00070007,
    kownerPrivileges = 0x00000007
};
```

Constants

`kioACAccessOwnerBit`

Indicates that the user is the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerMask`

The user is the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessBlankAccessBit`

Indicates that the directory has blank access privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioACAccessBlankAccessMask`
The directory has blank access privileges. A directory with blank access privileges set ignores the other access privilege bits and uses the access privilege bits of its parent directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserWriteBit`
Indicates that the user has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserWriteMask`
The user has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserReadBit`
Indicates that the user has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserReadMask`
The user has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserSearchBit`
Indicates that the user has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessUserSearchMask`
The user has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessEveryoneWriteBit`
Indicates that everyone has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessEveryoneWriteMask`
Everyone has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessEveryoneReadBit`
Indicates that everyone has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioACAccessEveryoneReadMask`
Everyone has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessEveryoneSearchBit`
Indicates that everyone has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessEveryoneSearchMask`
Everyone has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupWriteBit`
Indicates that the group has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupWriteMask`
The group has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupReadBit`
Indicates that the group has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupReadMask`
The group has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupSearchBit`
Indicates that the group has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessGroupSearchMask`
The group has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerWriteBit`
Indicates that the owner has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioACAccessOwnerWriteMask`
The owner has write privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerReadBit`
Indicates that the owner has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerReadMask`
The owner has read privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerSearchBit`
Indicates that the owner has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACAccessOwnerSearchMask`
The owner has search privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kfullPrivileges`
Indicates that everyone, including the owner, have all privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kownerPrivileges`
Indicates that only the owner has all privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
See `AccessParam` (page 180).

File Attribute Constants

Define file and directory attributes returned by the `PBGetCatInfoSync` and `PBGetCatInfoAsync` functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kioFlAttribLockedBit = 0,
    kioFlAttribLockedMask = 0x01,
    kioFlAttribResOpenBit = 2,
    kioFlAttribResOpenMask = 0x04,
    kioFlAttribDataOpenBit = 3,
    kioFlAttribDataOpenMask = 0x08,
    kioFlAttribDirBit = 4,
    kioFlAttribDirMask = 0x10,
    ioDirFlg = 4,
    ioDirMask = 0x10,
    kioFlAttribCopyProtBit = 6,
    kioFlAttribCopyProtMask = 0x40,
    kioFlAttribFileOpenBit = 7,
    kioFlAttribFileOpenMask = 0x80,
    kioFlAttribInSharedBit = 2,
    kioFlAttribInSharedMask = 0x04,
    kioFlAttribMountedBit = 3,
    kioFlAttribMountedMask = 0x08,
    kioFlAttribSharePointBit = 5,
    kioFlAttribSharePointMask = 0x20
};
```

Constants

`kioFlAttribLockedBit`

Indicates that the file or directory is locked. Use the functions `PBHSetFLockSync` and `PBHSetFLockAsync` to lock a file or directory. Use the functions `PBHRstFLockSync` and `PBHRstFLockAsync` to unlock a file or directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribLockedMask`

Tests if the file or directory is locked.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribResOpenBit`

Indicates that the resource fork is open. On Mac OS X, this bit is not set if the resource fork of the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribResOpenMask`

Tests if the resource fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDataOpenBit`

Indicates that the data fork is open. On Mac OS X, this bit is not set if the data fork of the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioFlAttribDataOpenMask`
Tests if the data fork is open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDirBit`
Indicates that this is a directory, not a file. This bit is always clear for files, and is always set for directories.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribDirMask`
Tests if this is a directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`ioDirFlg`
Indicates that this is a directory; this is the old name of the `kioFlAttribDirBit`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`ioDirMask`
Tests if this is a directory; this is the old name of the `kioFlAttribDirMask`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribCopyProtBit`
Indicates that the file is "copy-protected" by the AppleShare server.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribCopyProtMask`
Tests if the file is "copy-protected" by the AppleShare server.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribFileOpenBit`
Indicates that the file is open. This bit is set if either the data or the resource fork are open. On Mac OS X, this bit is not set if the file has been opened by a process other than the process making the call to `PBHGetCatInfo` or `PBHGetFInfo`.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribFileOpenMask`
Tests if the file is open. The file is open if either the data or the resource fork are open.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribInSharedBit`
Indicates that the directory is within a shared area of the directory hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioFlAttribInSharedMask`
Tests if the directory is within a shared area of the directory hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribMountedBit`
Indicates that the directory is a share point that is mounted by a user.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribMountedMask`
Tests if the directory is a share point that is mounted by a user.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribSharePointBit`
Indicates that the directory is a share point.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioFlAttribSharePointMask`
Tests if the directory is a share point.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioFlAttrib` fields of the `HFileInfo` (page 235) and `DirInfo` (page 195) structures returned by the functions `PBGetCatInfoSync` and `PBGetCatInfoAsync`.

File Operation Options

Flags you can use to specify how to perform a file operation.

```
enum {
    kFSFileOperationDefaultOptions = 0,
    kFSFileOperationOverwrite = 0x01,
    kFSFileOperationSkipSourcePermissionErrors = 0x02,
    kFSFileOperationDoNotMoveAcrossVolumes = 0x04,
    kFSFileOperationSkipPreflight = 0x08
};
```

Constants
`kFSFileOperationDefaultOptions`
Use the following default options:

- If the destination directory contains an object with the same name as a source object, abort the operation.
- If a source object cannot be read, abort the operation.
- If asked to move an object across volume boundaries, perform the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSFileOperationOverwrite`
If the destination directory contains an object with the same name as a source object, overwrite the destination object.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationSkipSourcePermissionErrors`
If a source object cannot be read, skip the object and continue the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationDoNotMoveAcrossVolumes`
If asked to move an object across volume boundaries, abort the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSFileOperationSkipPreflight`
Skip the preflight stage for a directory move or copy operation. This option limits the status information that can be returned during the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
These flags may be passed to any of the functions that initiate a file operation. For more information, see "Copying and Moving Objects Using Asynchronous High-Level File Operations" (page 21).

File Operation Stages

Constants used by the File Manager to indicate the current stage of an asynchronous file operation.

```
typedef UInt32 FSFileOperationStage;
enum {
    kFSOperationStageUndefined = 0,
    kFSOperationStagePreflighting = 1,
    kFSOperationStageRunning = 2,
    kFSOperationStageComplete = 3
};
```

Constants
`kFSOperationStageUndefined`
The File Manager has not started the file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationStagePreflighting`
The File Manager is performing tasks such as calculating the sizes and number of objects in the operation, and checking to make sure there is enough space on the destination volume to complete the operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSOperationStageRunning`
The File Manager is copying or moving the file or directory.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationStageComplete`
The file operation is complete.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
These constants are passed back to your file operation status callback function. For more information, see "File Operation Callbacks" (page 174). You can also get the current stage of a file operation by calling a status accessor function such as `FSFileOperationCopyStatus` (page 61).

File Operation Status Dictionary Keys

Keys used to determine the status of a file operation as reported in a status dictionary.

```
const CFStringRef kFSOperationTotalBytesKey;
const CFStringRef kFSOperationBytesCompleteKey;
const CFStringRef kFSOperationBytesRemainingKey;
const CFStringRef kFSOperationTotalObjectsKey;
const CFStringRef kFSOperationObjectsCompleteKey;
const CFStringRef kFSOperationObjectsRemainingKey;
const CFStringRef kFSOperationTotalUserVisibleObjectsKey;
const CFStringRef kFSOperationUserVisibleObjectsCompleteKey;
const CFStringRef kFSOperationUserVisibleObjectsRemainingKey;
const CFStringRef kFSOperationThroughputKey;
```

Constants
`kFSOperationTotalBytesKey`
The value for this key is a CFNumber that represents the total number of bytes that will be moved or copied by this file operation. This value is not available for a directory operation if the `kFSFileOperationSkipPreflight` (page 304) option flag is specified.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationBytesCompleteKey`
The value for this key is a CFNumber that represents the total number of bytes that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationBytesRemainingKey`
The value for this key is a CFNumber that represents the total number of bytes that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSOperationTotalObjectsKey`
The value for this key is a CFNumber that represents the total number of objects that will be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationObjectsCompleteKey`
The value for this key is a CFNumber that represents the total number of objects that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationObjectsRemainingKey`
The value for this key is a CFNumber that represents the total number of objects that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationTotalUserVisibleObjectsKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that will be moved or copied by this file operation. In general, an object is user-visible if it is displayed in a Finder window. For example, a package is counted as a single user-visible object even though it typically contains many other objects.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationUserVisibleObjectsCompleteKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that have already been moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationUserVisibleObjectsRemainingKey`
The value for this key is a CFNumber that represents the total number of user-visible objects that remain to be moved or copied by this file operation.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSOperationThroughputKey`
The value for this key is a CFNumber that represents the current throughput of this file operation in bytes per second.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Discussion
The status dictionary for a file operation is passed back to your status callback function. For more information, see "File Operation Callbacks" (page 174). You can also get the status dictionary for a file operation by calling a status accessor function such as `FSFileOperationCopyStatus` (page 61).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FNMessage

```
typedef UInt32 FNMessage;
enum {
    kFNDirectoryModifiedMessage = 1
};
```

Constants

`kFNDirectoryModifiedMessage`

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Foreign Privilege Model Constant

Identifies the A/UX privilege model.

```
enum {
    fsUnixPriv = 1
};
```

Constants

`fsUnixPriv`

Represents a volume that supports the A/UX privilege model.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion

Used in the `vMForeignPrivID` field of the `GetVolParmsInfoBuffer` (page 233).

Group ID Constant

```
enum {
    knoGroup = 0
};
```

Constants

`knoGroup`

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Icon Size Constants

Specify the sizes of the desktop database icon types.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kLargeIconSize = 256,
    kLarge4BitIconSize = 512,
    kLarge8BitIconSize = 1024,
    kSmallIconSize = 64,
    kSmall4BitIconSize = 128,
    kSmall8BitIconSize = 256
};
```

Constants

`kLargeIconSize`
Large black-and-white icon with mask. Corresponding resource type: '`ICN#`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kLarge4BitIconSize`
Large 4-bit color icon. Corresponding resource type: '`icl4`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kLarge8BitIconSize`
Large 8-bit color icon. Corresponding resource type: '`icl8`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmallIconSize`
Small black-and-white icon with mask. Corresponding resource type: '`ics#`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmall4BitIconSize`
Small 4-bit color icon. Corresponding resource type: '`ics4`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kSmall8BitIconSize`
Small 8-bit color icon. Corresponding resource type: '`ics8`'.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

These constants indicate the amount of storage you should allocate for the icon data for each of the icon types specified by the "Icon Type Constants" (page 308). The desktop database functions which set or retrieve icon data–namely, `PBDTAddIconSync`, `PBDTAddIconAsync`, `PBDTGetIconSync`, `PBDTGetIconAsync`, `PBDTGetIconInfoSync`, and `PBDTGetIconInfoAsync` –expect a pointer to the the storage in the `ioDTBuffer` field of the `DTPBRec` (page 199) parameter block and the appropriate constant in the `ioDTReqCount` field.

Icon Type Constants

Specify the icon types for the desktop database.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kLargeIcon = 1,
    kLarge4BitIcon = 2,
    kLarge8BitIcon = 3,
    kSmallIcon = 4,
    kSmall4BitIcon = 5,
    kSmall8BitIcon = 6,
    kicnsIconFamily = 239
};
```

Constants

`kLargeIcon`
Large black-and-white icon with mask. Corresponding resource type: '`ICN#`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kLarge4BitIcon`
Large 4-bit color icon. Corresponding resource type: '`icl4`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kLarge8BitIcon`
Large 8-bit color icon. Corresponding resource type: '`icl8`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmallIcon`
Small black-and-white icon with mask. Corresponding resource type: '`ics#`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmall4BitIcon`
Small 4-bit color icon. Corresponding resource type: '`ics4`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kSmall8BitIcon`
Small 8-bit color icon. Corresponding resource type: '`ics8`'.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

`kicnsIconFamily`
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
These constants are used in the `ioIconType` field of the `DTPBRec` (page 199) parameter block.

Invalid Volume Reference Constant

Represents an invalid volume reference number.

```
enum {
    kFSInvalidVolumeRefNum = 0
};
```

Constants
`kFSInvalidVolumeRefNum`
Invalid volume reference number.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Iterator Flags

Indicate whether an iterator iterates over subtrees or just the immediate children of the container.

```
enum {
    kFSIterateFlat = 0,
    kFSIterateSubtree = 1,
    kFSIterateDelete = 2,
    kFSIterateReserved = 0xFFFFFFFC
};
typedef OptionBits FSIteratorFlags;
```

Constants
`kFSIterateFlat`
Iterate over the immediate children of the container only.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateSubtree`
Iterate over the entire subtree rooted at the container.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateDelete`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kFSIterateReserved`
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

kAsyncMountInProgress

```
enum {
    kAsyncMountInProgress = 1,
    kAsyncMountComplete = 2,
    kAsyncUnmountInProgress = 3,
    kAsyncUnmountComplete = 4,
    kAsyncEjectInProgress = 5,
    kAsyncEjectComplete = 6
};
```

Constants

`kAsyncMountInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncMountComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncUnmountInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncUnmountComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncEjectInProgress`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

`kAsyncEjectComplete`
Available in Mac OS X v10.2 and later.
Declared in `Files.h`.

Notification Subscription Options

Options that can be specified at subscription time.

```
enum {
    kFNNoImplicitAllSubscription = (1 << 0),
    kFNNotifyInBackground = (1 << 1)
};
```

Constants

`kFNNoImplicitAllSubscription`
Specify this option if you do not want to receive notifications on this subscription when `FNNotifyAll` is called. By default, any subscription is also implicitly a subscription to wildcard notifications.
Available in Mac OS X v10.1 and later.
Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFNNotifyInBackground`
Specify this option if you want to receive notifications on this subscription when your application is in background. By default, notifications will be coalesced and and delivered when your application becomes foreground.

Available in Mac OS X v10.3 and later.

Declared in `Files.h`.

kHFSCatalogNodeIDsReusedBit

```
enum {
    kHFSCatalogNodeIDsReusedBit = 12,
    kHFSCatalogNodeIDsReusedMask = 1 << kHFSCatalogNodeIDsReusedBit
};
```

Constants

`kHFSCatalogNodeIDsReusedBit`
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `HFSVolumes.h`.

`kHFSCatalogNodeIDsReusedMask`
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared in `HFSVolumes.h`.

Large Volume Constants

```
enum {
    kWidePosOffsetBit = 8,
    kUseWidePositioning = (1 << kWidePosOffsetBit),
    kMaximumBlocksIn4GB = 0x007FFFFF
};
```

Constants

`kWidePosOffsetBit`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kUseWidePositioning`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kMaximumBlocksIn4GB`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Mapping Code Constants

Specify the type of object to map or return.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kOwnerID2Name = 1,
    kGroupID2Name = 2,
    kOwnerName2ID = 3,
    kGroupName2ID = 4,
    kReturnNextUser = 1,
    kReturnNextGroup = 2,
    kReturnNextUG = 3
};
```

Constants

`kOwnerID2Name`
Map a user ID to the user name. Used with the `PBHMapIDSync` or `PBHMapIDAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kGroupID2Name`
Map a group ID to the group name. Used with the `PBHMapIDSync` or `PBHMapIDAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kOwnerName2ID`
Map a user name to the user ID. Used with the `PBHMapNameSync` or `PBHMapNameAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kGroupName2ID`
Map a group name to the group ID. Used with the `PBHMapNameSync` or `PBHMapNameAsync` functions.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextUser`
Return the next user entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextGroup`
Return the next group entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kReturnNextUG`
Return the next user or group entry.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

These constants are used in the `ioObjType` field of the `ObjParam` (page 251) parameter block. The first four constants are passed to the `PBHMapIDSync`, `PBHMapIDAsync`, `PBHMapNameSync`, and `PBHMapNameAsync` functions to specify the mapping to be performed. The last three constants are passed to the `PBGetUGEntrySync` or `PBGetUGEntryAsync` functions to specify the type of object to be returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Path Conversion Options

Specify how a pathname is converted to an `FSRef` structure by the function `FSPathMakeRefWithOptions` (page 93).

```
enum {
    kFSPathMakeRefDefaultOptions = 0,
    kFSPathMakeRefDoNotFollowLeafSymlink = 0x01
};
```

Constants

`kFSPathMakeRefDefaultOptions`

Use the default options.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

`kFSPathMakeRefDoNotFollowLeafSymlink`

When converting a path that refers to a symbolic link, do not follow the link. The new `FSRef` should refer to the link itself.

Available in Mac OS X v10.4 and later.

Declared in `Files.h`.

Position Mode Constants

Together with an offset, specify a position within a fork.

```
enum {
    fsAtMark = 0,
    fsFromStart = 1,
    fsFromLEOF = 2,
    fsFromMark = 3
};
```

Constants

`fsAtMark`

The starting point is the access path's current position. The offset is ignored.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsFromStart`

The starting point is offset bytes from the start of the fork. The offset must be non-negative.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsFromLEOF`

The starting point is offset bytes from the logical end of the fork. The offset must not be positive.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`fsFromMark`
The starting point is offset bytes from the access path's current position. The offset may be positive or negative.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioPosMode` and `positionMode` fields and parameters of the HFS and HFS Plus file access functions. These functions include those for reading from and writing to files or forks, changing the current position within a file or fork, changing the size of a file or fork, and allocating space to a file or fork.

For the `FSReadFork` and `FSWriteFork` calls, you may also add either of the `pleaseCacheMask` or `noCacheMask` constants to hint whether the data should be cached or not. See "Cache Constants" (page 275).

Root Directory Constants

Specify the directory IDs of the root directory of a volume and its parent.

```
enum {
    fsRtParID = 1,
    fsRtDirID = 2
};
```

Constants
`fsRtParID`
Represents the directory ID of the root directory's parent directory. The root directory has no parent this constant is used when specifying the root directory to functions which require the parent directory ID to identify directories.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`fsRtDirID`
Represents the directory ID of the volume's root directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

User ID Constants

Specify basic user IDs for shared directories.

```
enum {
    knoUser = 0,
    kadministratorUser = 1
};
```

Constants
`knoUser`
Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kadministratorUser`

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

User Privileges Constants

Specify the user privileges for a directory on a remote volume.

```
enum {
    kioACUserNoSeeFolderBit = 0,
    kioACUserNoSeeFolderMask = 0x01,
    kioACUserNoSeeFilesBit = 1,
    kioACUserNoSeeFilesMask = 0x02,
    kioACUserNoMakeChangesBit = 2,
    kioACUserNoMakeChangesMask = 0x04,
    kioACUserNotOwnerBit = 7,
    kioACUserNotOwnerMask = 0x80
};
```

Constants

`kioACUserNoSeeFolderBit`

Set if the user does not have “See Folders” privileges. Without “See Folders” privileges, the user cannot see other directories in the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFolderMask`

Tests if the user has “See Folders” privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFilesBit`

Set if the user does not have “See Files” privileges. Without “See Files” privileges, the user cannot open documents or applications in the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoSeeFilesMask`

Tests if the user has “See Files” privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoMakeChangesBit`

Set if the user does not have “Make Changes” privileges. Without “Make Changes” privileges, the user cannot create, modify, rename, or delete any file or directory within the specified directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNoMakeChangesMask`

Tests if the user has “Make Changes” privileges.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioACUserNotOwnerBit`

Set if the user is not the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioACUserNotOwnerMask`

Tests whether the user is the owner of the directory.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion

These constants are used in the `ioACUser` field of the `HFileInfo` (page 235) and `DirInfo` (page 195) structures returned by the `PBGetCatInfoSync` and `PBGetCatInfoAsync` functions.

Volume Attribute Constants

Bit position constants that specify volume attributes.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    bLimitFCBs = 31,
    bLocalWList = 30,
    bNoMiniFndr = 29,
    bNoVNEdit = 28,
    bNoLclSync = 27,
    bTrshOffLine = 26,
    bNoSwitchTo = 25,
    bNoDeskItems = 20,
    bNoBootBlks = 19,
    bAccessCntl = 18,
    bNoSysDir = 17,
    bHasExtFSVol = 16,
    bHasOpenDeny = 15,
    bHasCopyFile = 14,
    bHasMoveRename = 13,
    bHasDesktopMgr = 12,
    bHasShortName = 11,
    bHasFolderLock = 10,
    bHasPersonalAccessPrivileges = 9,
    bHasUserGroupList = 8,
    bHasCatSearch = 7,
    bHasFileIDs = 6,
    bHasBTreeMgr = 5,
    bHasBlankAccessPrivileges = 4,
    bSupportsAsyncRequests = 3,
    bSupportsTrashVolumeCache = 2
};
enum {
    bHasDirectIO = 1
};
```

Constants

`bLimitFCBs`

The Finder limits the number of file control blocks used during copying to 8 instead of 16.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bLocalWList`

The Finder uses the returned shared volume handle for its local window list.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoMiniFndr`

Reserved; always set to 1.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoVNEdit`

This volume's name cannot be edited.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bNoLclSync`
Don't let the Finder change the modification date.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bTrshOffLine`
Any time this volume goes offline, it is zoomed to the Trash and unmounted.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoSwitchTo`
The Finder will not switch launch to any application on this volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoDeskItems`
Don't place objects in this volume on the Finder desktop.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bNoBootBlks`
This volume is not a startup volume. The Startup menu item is disabled. Boot blocks are not copied during copy operations.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bAccessCntl`
This volume supports AppleTalk AFP access-control interfaces. The following functions are supported:

- `PBHGetLogInInfoSync`
- `PBHGetLogInInfoAsync`
- `PBHGetDirAccessSync`
- `PBHGetDirAccessAsync`
- `PBHSetDirAccessSync`
- `PBHSetDirAccessAsync`
- `PBHMapIDSync`
- `PBHMapIDAsync`
- `PBHMapNameSync`
- `PBHMapNameAsync`

Special folder icons are used. The Access Privileges menu command is enabled for disk and folder items. The `ioFlAttrib` field of the parameter block passed to the `PBGetCatInfoSync` and `PBGetCatInfoSync` functions is assumed to be valid.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bNoSysDir`
This volume doesn't support a system directory. Do not switch launch to this volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasExtFSVol`
This volume is an external file system volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasOpenDeny`
This volume supports the `PBHOpenDenySync` , `PBHOpenDenyAsync`, `PBHOpenRFDenySync` and `PBHOpenRFDenyAsync` functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasCopyFile`
This volume supports the `PBHCopyFileSync` and `PBHCopyFileAsync` functions, which is used in copy and duplicate operations if both source and destination volumes have the same server address.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasMoveRename`
This volume supports the `PBHMoveRenameSync` and `PBHMoveRenameAsync` functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasDesktopMgr`
This volume supports all of the desktop functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasShortName`
This volume supports AFP short names.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasFolderLock`
Folders on the volume can be locked, and so they cannot be deleted or renamed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasPersonalAccessPrivileges`
This volume has local file sharing enabled.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`bHasUserGroupList`
This volume supports the Users and Groups file and thus the AFP privilege functions.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`bHasCatSearch`
: This volume supports the `PBCatSearchSync` and `PBCatSearchAsync` functions.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasFileIDs`
: This volume supports the file ID functions, including the `PBExchangeFilesSync` and `PBExchangeFilesAsync` functions.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasBTreeMgr`
: Reserved for internal use.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bHasBlankAccessPrivileges`
: This volume supports inherited access privileges for folders (blank access privileges).

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bSupportsAsyncRequests`
: This volume correctly handles asynchronous requests at any time.

  Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

`bSupportsTrashVolumeCache`
: Available in Mac OS X v10.0 and later.

  Declared in `Files.h`.

Discussion
These constants correspond to bit positions in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 233) structure returned by the `PBHGetVolParmsSync` (page 518) and `PBHGetVolParmsAsync` (page 517) functions.

Volume Control Block Flags

Used in the `vcbFlags` field of a volume control block to specify information about a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kVCBFlagsIdleFlushBit = 3,
    kVCBFlagsIdleFlushMask = 0x0008,
    kVCBFlagsHFSPlusAPIsBit = 4,
    kVCBFlagsHFSPlusAPIsMask = 0x0010,
    kVCBFlagsHardwareGoneBit = 5,
    kVCBFlagsHardwareGoneMask = 0x0020,
    kVCBFlagsVolumeDirtyBit = 15,
    kVCBFlagsVolumeDirtyMask = 0x8000
};
```

Constants

`kVCBFlagsIdleFlushBit`
Indicates that the volume should be flushed at idle time.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsIdleFlushMask`
Flushes the volume at idle time.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHFSPlusAPIsBit`
Indicates that the volume directly implements the HFS Plus APIs (rather than emulating them).
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHFSPlusAPIsMask`
The volume directly implements the HFS Plus APIs.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHardwareGoneBit`
Indicates that the disk driver returned a `hardwareGoneErr` in response to a read or write call.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsHardwareGoneMask`
Tests if the disk driver returned a `hardwareGoneErr` in response to a read or write call.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsVolumeDirtyBit`
Indicates that the volume information has changed since the last time the volume was flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

`kVCBFlagsVolumeDirtyMask`
The volume has changed since the last time the volume was flushed.
Available in Mac OS X v10.0 and later.
Declared in `Files.h`.

Discussion

See `VCB` (page 254) for a description of the volume control block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Volume Information Attribute Constants

Define volume attributes returned by the functions `PBHGetVInfoSync`, `PBHGetVInfoAsync`, `PBXGetVolInfoSync`, and `PBXGetVolInfoAsync`.

```
enum {
    kioVAtrbDefaultVolumeBit = 5,
    kioVAtrbDefaultVolumeMask = 0x0020,
    kioVAtrbFilesOpenBit = 6,
    kioVAtrbFilesOpenMask = 0x0040,
    kioVAtrbHardwareLockedBit = 7,
    kioVAtrbHardwareLockedMask = 0x0080,
    kioVAtrbSoftwareLockedBit = 15,
    kioVAtrbSoftwareLockedMask = 0x8000
};
```

Constants

`kioVAtrbDefaultVolumeBit`
Indicates that the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbDefaultVolumeMask`
Tests if the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbFilesOpenBit`
Indicates that there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbFilesOpenMask`
Tests if there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbHardwareLockedBit`
Indicates that the volume is locked by a hardware setting. On Mac OS X, the File Manager only sets the software locked bit for CDs and other read-only media; it does not set the hardware locked bit.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbHardwareLockedMask`
Tests if the volume is locked by a hardware setting.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kioVAtrbSoftwareLockedBit`
Indicates that the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kioVAtrbSoftwareLockedMask`
Tests if the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used in the `ioVAtrb` field of the `HVolumeParam` (page 245) parameter block returned by the `PBHGetVInfoSync` (page 450) and `PBHGetVInfoAsync` (page 447) functions, and in the `ioVAtrb` field of the `XVolumeParam` (page 268) parameter block returned by the `PBXGetVolInfoSync` (page 497) and `PBXGetVolInfoAsync` (page 494) functions.

Volume Information Bitmap Constants

Indicate what volume information to set or retrieve.

```
enum {
    kFSVolInfoNone = 0x0000,
    kFSVolInfoCreateDate = 0x0001,
    kFSVolInfoModDate = 0x0002,
    kFSVolInfoBackupDate = 0x0004,
    kFSVolInfoCheckedDate = 0x0008,
    kFSVolInfoFileCount = 0x0010,
    kFSVolInfoDirCount = 0x0020,
    kFSVolInfoSizes = 0x0040,
    kFSVolInfoBlocks = 0x0080,
    kFSVolInfoNextAlloc = 0x0100,
    kFSVolInfoRsrcClump = 0x0200,
    kFSVolInfoDataClump = 0x0400,
    kFSVolInfoNextID = 0x0800,
    kFSVolInfoFinderInfo = 0x1000,
    kFSVolInfoFlags = 0x2000,
    kFSVolInfoFSInfo = 0x4000,
    kFSVolInfoDriveInfo = 0x8000,
    kFSVolInfoGettableInfo = 0xFFFF,
    kFSVolInfoSettableInfo = 0x3004
};
```

Constants
`kFSVolInfoNone`
No volume information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoCreateDate`
Retrieve the creation date of the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoModDate`
Retrieve the date of the volume's last modification.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSVolInfoBackupDate`
Retrieve or set the date of the volume's last backup.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoCheckedDate`
Retrieve the date that the volume was last checked for consistency.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFileCount`
Retrieve the number of files on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDirCount`
Retrieve the number of directories on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoSizes`
Retrieve the total number of bytes on the volume and the number of unused bytes on the volume (in the `totalBytes` and `freeBytes` fields).

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoBlocks`
Retrieve the block information: the block size, the number of total blocks on the volume, and the number of free blocks on the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoNextAlloc`
Retrieve the address at which to start the next allocation.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoRsrcClump`
Retrieve the resource fork clump size.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDataClump`
Retrieve the data fork clump size.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoNextID`
Retrieve the next available catalog node ID.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`kFSVolInfoFinderInfo`
Retrieve or set the volume's Finder information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFlags`
Retrieve or set the volume's flags. See "Volume Information Flags" (page 326) for more information on the volume's flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoFSInfo`
Retrieve the filesystem ID and signature.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoDriveInfo`
Retrieve the drive information: the drive number and driver reference number.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoGettableInfo`
Retrieve all of the gettable information.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolInfoSettableInfo`
Set all of the settable information. Currently, this is the backup date, Finder information, and flags.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
These constants are used with the `FSVolumeInfoBitmap` (page 231) data type to indicate what volume information to set or retrieve with the functions `FSSetVolumeInfo` (page 103) and `FSGetVolumeInfo` (page 74) , and their corresponding parameter block calls.

Volume Information Flags

Used by the `FSVolumeInfo` structure to specify characteristics of a volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
enum {
    kFSVolFlagDefaultVolumeBit = 5,
    kFSVolFlagDefaultVolumeMask = 0x0020,
    kFSVolFlagFilesOpenBit = 6,
    kFSVolFlagFilesOpenMask = 0x0040,
    kFSVolFlagHardwareLockedBit = 7,
    kFSVolFlagHardwareLockedMask = 0x0080,
    kFSVolFlagSoftwareLockedBit = 15,
    kFSVolFlagSoftwareLockedMask = 0x8000
};
```

Constants

`kFSVolFlagDefaultVolumeBit`
Set if the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagDefaultVolumeMask`
Indicates that the volume is the default volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagFilesOpenBit`
Set if there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagFilesOpenMask`
Indicates that there are open files or iterators.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagHardwareLockedBit`
Set if the volume is locked by a hardware setting. On Mac OS X, the File Manager only sets the software locked bit for CDs and other read-only media; it does not set the hardware locked bit.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagHardwareLockedMask`
Indicates that the volume is locked by a hardware setting.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagSoftwareLockedBit`
Set if the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`kFSVolFlagSoftwareLockedMask`
Indicates that the volume is locked by software.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
See the `flags` field of the `FSVolumeInfo` (page 228) structure.

Volume Mount Flags

Define flags used by the volume mounting information structures.

```
enum {
    volMountNoLoginMsgFlagBit = 0,
    volMountNoLoginMsgFlagMask = 0x0001,
    volMountExtendedFlagsBit = 7,
    volMountExtendedFlagsMask = 0x0080,
    volMountInteractBit = 15,
    volMountInteractMask = 0x8000,
    volMountChangedBit = 14,
    volMountChangedMask = 0x4000,
    volMountFSReservedMask = 0x00FF,
    volMountSysReservedMask = 0xFF00
};
```

Constants

`volMountNoLoginMsgFlagBit`
Indicates that any log-in message or greeting dialog will be suppressed.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountNoLoginMsgFlagMask`
Tells the file system to suppress any log-in message or greeting dialog.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountExtendedFlagsBit`
Indicates that the mounting information is a `AFPXVolMountInfo` record for AppleShare Client version 3.7 and later.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountExtendedFlagsMask`
Tells the file system that the mounting information is an `AFPXVolMountInfo` (page 185) record for AppleShare Client version 3.7 and later.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountInteractBit`
Indicates that it's safe for the file system to perform user interaction to mount the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountInteractMask`
Tells the file system that it's safe to perform user interaction to mount the volume.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`volMountChangedBit`
Indicates that the volume was mounted, but the volume mounting information record needs to be updated.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountChangedMask`
Tests if the volume mounting information record needs to be updated.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountFSReservedMask`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

`volMountSysReservedMask`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Files.h`.

Discussion
Bits 0-7 are defined by each file system for its own use; bits 8-15 are reserved for Apple system use. These constants are used in the `flags` fields of the `AFPVolMountInfo` (page 183), `AFPXVolMountInfo` (page 185) , and `VolumeMountInfoHeader` (page 259) structures.

Result Codes

The most common result codes returned by File Manager functions are listed below.

Result Code	Value	Description
`dirFulErr`	-33	File directory full. Available in Mac OS X v10.0 and later.
`dskFulErr`	-34	Disk or volume full. Available in Mac OS X v10.0 and later.
`nsvErr`	-35	Volume not found. Available in Mac OS X v10.0 and later.
`ioErr`	-36	I/O error. Available in Mac OS X v10.0 and later.
`bdNamErr`	-37	Bad filename or volume name. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
fnOpnErr	-38	File not open. Available in Mac OS X v10.0 and later.
eofErr	-39	Logical end-of-file reached. Available in Mac OS X v10.0 and later.
posErr	-40	Attempt to position mark before the start of the file. Available in Mac OS X v10.0 and later.
mFulErr	-41	Memory full (open) or file won't fit (load) Available in Mac OS X v10.0 and later.
tmfoErr	-42	Too many files open. Available in Mac OS X v10.0 and later.
fnfErr	-43	File or directory not found; incomplete pathname. Available in Mac OS X v10.0 and later.
wPrErr	-44	Volume is locked through hardware. Available in Mac OS X v10.0 and later.
fLckdErr	-45	File is locked. Available in Mac OS X v10.0 and later.
vLckdErr	-46	Volume is locked through software. Available in Mac OS X v10.0 and later.
fBsyErr	-47	One or more files are open File is busy Directory is not empty. Available in Mac OS X v10.0 and later.
dupFNErr	-48	Duplicate filename and version Destination file already exists File found instead of folder Available in Mac OS X v10.0 and later.
opWrErr	-49	File already open for writing. Available in Mac OS X v10.0 and later.
paramErr	-50	Invalid value passed in a parameter. Your application passed an invalid parameter for dialog options. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
rfNumErr	-51	Invalid reference number. Available in Mac OS X v10.0 and later.
gfpErr	-52	Error during GetFPos, PBGetFPosSync or PBGetFPosAsync. Available in Mac OS X v10.0 and later.
volOffLinErr	-53	Volume is offline. Available in Mac OS X v10.0 and later.
permErr	-54	Attempt to open locked file for writing Permissions error Available in Mac OS X v10.0 and later.
volOnLinErr	-55	Volume already online. Available in Mac OS X v10.0 and later.
nsDrvErr	-56	No such drive. Available in Mac OS X v10.0 and later.
noMacDskErr	-57	Not a Macintosh disk. Available in Mac OS X v10.0 and later.
extFSErr	-58	Volume belongs to an external file system. Available in Mac OS X v10.0 and later.
fsRnErr	-59	Problem during rename. Available in Mac OS X v10.0 and later.
badMDBErr	-60	Bad master directory block. Available in Mac OS X v10.0 and later.
wrPermErr	-61	Read/ write permission doesn't allow writing. Available in Mac OS X v10.0 and later.
lastDskErr	-64	Available in Mac OS X v10.0 and later.
noDriveErr	-64	Drive not installed. Available in Mac OS X v10.0 and later.
firstDskErr	-84	Available in Mac OS X v10.0 and later.
dirNFErr	-120	Directory not found or incomplete pathname. Available in Mac OS X v10.0 and later.
tmwdoErr	-121	Too many working directories open. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
badMovErr	-122	Attempt to move. Available in Mac OS X v10.0 and later.
wrgVolTypErr	-123	Volume does not support Desktop Manager Not an HFS volume Available in Mac OS X v10.0 and later.
volGoneErr	-124	Server volume has been disconnected. Available in Mac OS X v10.0 and later.
fsDSIntErr	-127	non-hardware internal file system error. Available in Mac OS X v10.0 and later.
fsmFFSNotFoundErr	-431	Foreign file system does not exist. Available in Mac OS X v10.0 and later.
fsmBusyFFSErr	-432	File system is busy, cannot be removed. Available in Mac OS X v10.0 and later.
fsmBadFFSNameErr	-433	Name length not 1 <= length <= 31 Available in Mac OS X v10.0 and later.
fsmBadFSDLenErr	-434	FSD size incompatible with current FSM vers Available in Mac OS X v10.0 and later.
fsmDuplicateFSIDErr	-435	FSID already exists. Available in Mac OS X v10.0 and later.
fsmBadFSDVersionErr	-436	FSM version incompatible with FSD Available in Mac OS X v10.0 and later.
fsmNoAlternateStackErr	-437	no alternate stack for HFS CI Available in Mac OS X v10.0 and later.
fsmUnknownFSMMessageErr	-438	unknown message passed to FSM Available in Mac OS X v10.0 and later.
driverHardwareGoneErr	-503	disk driver's hardware was disconnected Available in Mac OS X v10.0 and later.
fidNotFound	-1300	File ID not found Available in Mac OS X v10.0 and later.
fidExists	-1301	File ID already exists Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
notAFileErr	-1302	Specified file is a directory Available in Mac OS X v10.0 and later.
diffVolErr	-1303	Files on different volumes Available in Mac OS X v10.0 and later.
catChangedErr	-1304	Catalog has changed and catalog position record may be invalid Available in Mac OS X v10.0 and later.
sameFileErr	-1306	Can't exchange a file with itself Available in Mac OS X v10.0 and later.
badFidErr	-1307	File ID is dangling or doesn't match with the file number Available in Mac OS X v10.0 and later.
notARemountErr	-1308	_Mount allows only remounts and doesn't get one Available in Mac OS X v10.0 and later.
fileBoundsErr	-1309	File's EOF, offset, mark or size is too big Available in Mac OS X v10.0 and later.
fsDataTooBigErr	-1310	File or volume is too big for system Available in Mac OS X v10.0 and later.
volVMBusyErr	-1311	Can't eject because volume is in use by VM Available in Mac OS X v10.0 and later.
badFCBErr	-1327	FCBRecPtr is not valid Available in Mac OS X v10.0 and later.
errFSUnknownCall	-1400	Selector is not recognized by this file system Available in Mac OS X v10.0 and later.
errFSBadFSRef	-1401	An FSRef parameter was invalid. There are several possible causes: The parameter was not optional, but the pointer was NULL. The volume reference number contained within the FSRef does not match a currently mounted volume. This can happen if the volume was unmounted after the FSRef was created. Some other private field inside the FSRef contains a value that could never be valid. If the field value could be valid, but doesn't happen to match the existing volume or in-memory structures, a "not found" error would be returned instead. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errFSBadForkName	-1402	A supplied fork name was invalid (i.e., was syntactically illegal for the given volume). For example, the fork name might contain characters that cannot be stored on the given volume (such as a colon on HFS volumes). Some volume formats do not store fork names in Unicode. These volume formats will attempt to convert the Unicode name to the kind of encoding used by the volume format. If the name could not be converted, errFSBadForkName is returned. Some volume formats only support a limited set of forks, such as the data and resource forks on HFS volumes. For those volumes, if any other fork name is passed, this error is returned. Available in Mac OS X v10.0 and later.
errFSBadBuffer	-1403	A non-optional buffer pointer was NULL , or its size was invalid for the type of data it was expected to contain. In a protected memory system, this could also mean the buffer space is not part of the address space for the calling process. Available in Mac OS X v10.0 and later.
errFSBadForkRef	-1404	A file reference number does not correspond to a fork opened with the FSOpenFork, PBOpenForkSync , or PBOpenForkAsync functions. This could be because that fork has already been closed. Or, you may have passed a reference number created with older APIs (e.g., by the PBHOpenDF functions). A value of zero is never a valid file reference number. Available in Mac OS X v10.0 and later.
errFSBadInfoBitmap	-1405	A FSCatalogInfoBitmap or FSVolumeInfoBitmap has one or more reserved or undefined bits set. This error code can also be returned if a defined bit is set, but the corresponding FSCatalogInfo or FSVolumeInfo field cannot be operated on with that call (for example, trying to use FSSetCatalogInfo to set the valence of a directory). Available in Mac OS X v10.0 and later.
errFSMissingCatInfo	-1406	A FSCatalogInfo pointer is NULL , but is not optional. Or, the FSCatalogInfo is optional and NULL, but the corresponding FSCatalogInfoBitmap is not zero (that is,the bitmap says that one or more of the FSCatalogInfo fields is being passed, but the supplied pointer was NULL). Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errFSNotAFolder	-1407	A parameter was expected to identify a folder, but it identified some other kind of object (e.g., a file) instead. This implies that the specified object exists, but is of the wrong type. For example, one of the parameters to FSCreateFileUnicode is an FSRef of the directory where the file will be created; if the FSRef actually refers to a file, this error is returned. Available in Mac OS X v10.0 and later.
errFSForkNotFound	-1409	An attempt to specify a fork of a given file or directory, but that particular fork does not exist. Available in Mac OS X v10.0 and later.
errFSNameTooLong	-1410	A file or fork name was too long. This means that the given name could never exist; this is different from a “file not found” or errFSForkNotFound error. Available in Mac OS X v10.0 and later.
errFSMissingName	-1411	A required file or fork name parameter was a NULL pointer, or the length of a filename was zero. Available in Mac OS X v10.0 and later.
errFSBadPosMode	-1412	Reserved or invalid bits in a positionMode field were set. For example, the FSReadFork call does not support newline mode, so setting the newline bit or a newline character in the positionMode parameter would cause this error. Available in Mac OS X v10.0 and later.
errFSBadAllocFlags	-1413	Reserved or invalid bits were set in an FSAllocationFlags parameter. Available in Mac OS X v10.0 and later.
errFSNoMoreItems	-1417	There are no more items to return when enumerating a directory or searching a volume. Note that FSCatalogSearch returns this error, whereas the PBCatSearch functions would return eofErr. Available in Mac OS X v10.0 and later.
errFSBadItemCount	-1418	The maximumObjects parameter to FSGetCatalogInfoBulk or FSCatalogSearch was zero. Available in Mac OS X v10.0 and later.
errFSBadSearchParams	-1419	The search criteria to FSCatalogSearch are invalid or inconsistent. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errFSRefsDifferent	-1420	The two FSRef structures passed to FSCompareFSRefs are for different files or directories. Note that a volume format may be able to compare the FSRef structures without searching for the files or directories, so this error may be returned even if one or both of the FSRef structures refers to non-existent objects. Available in Mac OS X v10.0 and later.
errFSForkExists	-1421	An attempt to create a fork, but that fork already exists. Available in Mac OS X v10.0 and later.
errFSBadIteratorFlags	-1422	The flags passed to FSOpenIterator are invalid. Available in Mac OS X v10.0 and later.
errFSIteratorNotFound	-1423	The value of an FSIterator parameter does not correspond to any currently open iterator. Available in Mac OS X v10.0 and later.
errFSIteratorNotSupported	-1424	The iterator flags or container of an FSIterator are not supported by that call. For example, in the initial release, the FSCatalogSearch call only supports an iterator whose container is in the volume's root directory and whose flags are kFSIterateSubtree (i.e., an iterator for the entire volume's contents). Similarly, in the initial release, FSGetCatalogInfoBulk only supports an iterator whose flags are kFSIterateFlat. Available in Mac OS X v10.0 and later.
errFSQuotaExceeded	-1425	The user's quota of disk blocks has been exhausted. Available in Mac OS X v10.2 and later.
afpAccessDenied	-5000	User does not have the correct access to the file Directory cannot be shared Available in Mac OS X v10.0 and later.
afpAuthContinue	-5001	Further information required to complete AFPLogin call. Available in Mac OS X v10.0 and later.
afpBadUAM	-5002	User authentication method is unknown. Available in Mac OS X v10.0 and later.
afpBadVersNum	-5003	Workstation is using an AFP version that the server doesn't recognize. Available in Mac OS X v10.0 and later.
afpBitmapErr	-5004	Bitmap contained bits undefined for call. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
afpCantMove	-5005	Move destination is offspring of source or root was specified. Available in Mac OS X v10.0 and later.
afpDenyConflict	-5006	Requested user permission not possible. Available in Mac OS X v10.0 and later.
afpDirNotEmpty	-5007	Cannot delete non-empty directory. Available in Mac OS X v10.0 and later.
afpDiskFull	-5008	Insufficient free space on volume for operation. Available in Mac OS X v10.0 and later.
afpEofError	-5009	Read beyond logical end-of-file. Available in Mac OS X v10.0 and later.
afpFileBusy	-5010	Cannot delete an open file. Available in Mac OS X v10.0 and later.
afpFlatVol	-5011	Cannot create directory on specified volume. Available in Mac OS X v10.0 and later.
afpItemNotFound	-5012	Unknown user name/ user ID or missing comment / APPL entry. Available in Mac OS X v10.0 and later.
afpLockErr	-5013	Some or all of requested range is locked by another user. Available in Mac OS X v10.0 and later.
afpMiscErr	-5014	Unexpected error encountered during execution. Available in Mac OS X v10.0 and later.
afpNoMoreLocks	-5015	No more ranges can be locked. Available in Mac OS X v10.0 and later.
afpNoServer	-5016	Server is not responding. Available in Mac OS X v10.0 and later.
afpObjectExists	-5017	Specified destination file or directory already exists. Available in Mac OS X v10.0 and later.
afpObjectNotFound	-5018	Specified file or directory does not exist. Available in Mac OS X v10.0 and later.
afpParmErr	-5019	A specified parameter was out of allowable range. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
afpRangeNotLocked	-5020	Specified range was not locked. Available in Mac OS X v10.0 and later.
afpRangeOverlap	-5021	Part of range is already locked. Available in Mac OS X v10.0 and later.
afpSessClosed	-5022	Session closed. Available in Mac OS X v10.0 and later.
afpUserNotAuth	-5023	User authentication failed (usually, password is not correct). Available in Mac OS X v10.0 and later.
afpCallNotSupported	-5024	Unsupported AFP call was made. Available in Mac OS X v10.0 and later.
afpObjectTypeErr	-5025	A directory exists with that name Directory not found Folder locking not supported by volume Object was a file, not a directory Available in Mac OS X v10.0 and later.
afpTooManyFilesOpen	-5026	Maximum open file count reached. Available in Mac OS X v10.0 and later.
afpServerGoingDown	-5027	Server is shutting down. Available in Mac OS X v10.0 and later.
afpCantRename	-5028	AFPRename cannot rename volume. Available in Mac OS X v10.0 and later.
afpDirNotFound	-5029	Unknown directory specified. Available in Mac OS X v10.0 and later.
afpIconTypeError	-5030	Icon size specified is different from existing icon size. Available in Mac OS X v10.0 and later.
afpVolLocked	-5031	Volume is read-only. Available in Mac OS X v10.0 and later.
afpObjectLocked	-5032	Object is M/R/D/W inhibited. Available in Mac OS X v10.0 and later.
afpContainsSharedErr	-5033	The directory contains a share point. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
afpIDNotFound	-5034	File ID not found. Available in Mac OS X v10.0 and later.
afpIDExists	-5035	File ID already exists. Available in Mac OS X v10.0 and later.
afpDiffVolErr	-5036	Available in Mac OS X v10.0 and later.
afpCatalogChanged	-5037	Catalog has changed and search cannot be resumed. Available in Mac OS X v10.0 and later.
afpSameObjectErr	-5038	Source and destination files are the same. Available in Mac OS X v10.0 and later.
afpBadIDErr	-5039	File ID not found. Available in Mac OS X v10.0 and later.
afpPwdSameErr	-5040	Someone tried to change their password to the same password on a mandatory password change. Available in Mac OS X v10.0 and later.
afpPwdTooShortErr	-5041	The password being set is too short: there is a minimum length that must be met or exceeded. Available in Mac OS X v10.0 and later.
afpPwdExpiredErr	-5042	Password has expired on server. Available in Mac OS X v10.0 and later.
afpInsideSharedErr	-5043	The directory is inside a shared directory. Available in Mac OS X v10.0 and later.
afpInsideTrashErr	-5044	The folder being shared is inside the trash folder OR the shared folder is being moved into the trash folder. Available in Mac OS X v10.0 and later.
afpPwdNeedsChangeErr	-5045	The password needs to be changed. Available in Mac OS X v10.0 and later.
afpPwdPolicyErr	-5046	Password does not conform to server's password policy. Available in Mac OS X v10.0 and later.
afpAlreadyLoggedInErr	-5047	User has been authenticated but is already logged in from another machine (and that's not allowed on this server). Available in Mac OS X v10.0 and later.
afpCallNotAllowed	-5048	Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Result Code	Value	Description
`afpBadDirIDType`	-5060	Not a fixed directory ID volume. Available in Mac OS X v10.0 and later.
`afpCantMountMoreSrvre`	-5061	Maximum number of volumes has been mounted. Available in Mac OS X v10.0 and later.
`afpAlreadyMounted`	-5062	Volume already mounted. Available in Mac OS X v10.0 and later.
`afpSameNodeErr`	-5063	Attempt to log on to a server running on the same machine. Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated File Manager Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.4

Allocate

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `FSAllocateFork` (page 43) instead.)

```
OSErr Allocate (
   FSIORefNum refNum,
   SInt32 *count
);
```

Parameters

refNum

The file reference number of the open file.

count

On input, a pointer to the number of additional bytes to allocate to the file. On return, a pointer to the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `Allocate` function adds the specified number of bytes to the file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, `Allocate` allocates the rest of the space on the volume and returns `dskFulErr` as its function result.

The `Allocate` function always attempts to allocate contiguous blocks. If the total number of requested bytes is unavailable, `Allocate` allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call `AllocContig` (page 342) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `Allocate` or `AllocContig` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

When the File Manager allocates (or deallocates) file blocks automatically, it always adds (or removes) blocks in clumps. The `Allocate` function allows you to add blocks in allocation blocks, which may be smaller than clumps.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 499) function, or by writing data to the file with the `FSWrite` (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 499) function, or one of the related parameter block calls, `PBSetEOFSync` (page 484) and `PBSetEOFAsync` (page 483).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the `Allocate` function does not return the number of bytes actually allocated. Your application should not rely on the value returned in the `count` parameter.

To determine the remaining space on a volume before calling `Allocate`, use the functions `PBXGetVolInfoSync` or `PBXGetVolInfoAsync`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

SampleUSBMIDIDriver

Declared In

`Files.h`

AllocContig

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `FSAllocateFork` (page 43) instead.)

```
OSErr AllocContig (
   FSVolumeRefNum refNum,
   SInt32 *count
);
```

Parameters

refNum

The file reference number of an open file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

count

On input, a pointer to the number of additional bytes to allocate to the file; on return, a pointer to the number of bytes allocated, rounded up to the nearest multiple of the allocation block size.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `AllocContig` function is identical to the `Allocate` (page 341) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, `AllocContig` does nothing and returns `dskFulErr` as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call `Allocate` (page 341) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `AllocContig` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

When the File Manager allocates (or deallocates) file blocks automatically, it always adds (or removes) blocks in clumps. The `AllocContig` function allows you to add blocks in allocation blocks, which may be smaller than clumps.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 499) function, or by writing data to the file with the `FSWrite` (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 499) function, or one of the related parameter block calls, `PBSetEOFSync` (page 484) and `PBSetEOFAsync` (page 483).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, `AllocContig` does not return 0 in the `count` parameter, so your application cannot rely upon this value.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

CatMove

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use `FSMoveObject` (page 82) instead.)

```
OSErr CatMove (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param oldName,
   SInt32 newDirID,
   ConstStr255Param newName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to move.

oldName

The existing name of the file or directory to move.

newDirID

If the `newName` parameter is empty, the directory ID of the destination directory; otherwise, the parent directory ID of the destination directory.

newName

The name of the destination directory. If a valid directory name is provided in this parameter, the destination directory's parent directory is specified in the `newDirID` parameter. However, you can specify an empty name for `newName`, in which case `newDirID` should be set to the directory ID of the destination directory.

It is usually simplest to specify the destination directory by passing its directory ID in the `newDirID` parameter and by setting `newName` to an empty name. To specify an empty name, set `newName` to `':'`.

Return Value

A result code. See "File Manager Result Codes" (page 329). This function returns `permErr` if called on a locked file.

Discussion

`CatMove` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk.

The `CatMove` function cannot move a file or directory to another volume (that is, the `vRefNum` parameter is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `HRename` (page 369).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 82) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 153) and `PBMoveObjectAsync` (page 152).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

DirCreate

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `FSCreateDirectoryUnicode` (page 52) instead.)

```
OSErr DirCreate (
   FSVolumeRefNum vRefNum,
   SInt32 parentDirID,
   ConstStr255Param directoryName,
   SInt32 *createdDirID
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

parentDirID

The directory ID of the parent directory. If the parent directory ID is 0 and the volume specified in the `vRefNum` parameter is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the `vRefNum` parameter is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant `fsRtDirID`(2) in this parameter.

directoryName

The name of the new directory.

createdDirID

On return, a pointer to the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The date and time of the new directory's creation and last modification are set to the current date and time.

To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 52) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 123) and `PBCreateDirectoryUnicodeAsync` (page 122).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSClose

Closes an open file. (Deprecated in Mac OS X v10.4. Use `FSCloseFork` (page 47) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSClose (
    FSIORefNum refNum
);
```

Parameters

refNum

The file reference number of the open file.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `FSClose` function removes the access path for the specified file and writes the contents of the volume buffer to the volume.

The `FSClose` function calls the `PBFlushFileSync` function internally to write the file's bytes onto the volume. To ensure that the file's catalog entry is updated, you should call `FlushVol` (page 502) after you call `FSClose`.

Special Considerations

Make sure that you do not call `FSClose` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

JPEG File Interchange Format

qtdataexchange

qtdataexchange.win

qtmoviefromprocs

qtmoviefromprocs.win

Declared In

`Files.h`

FSMakeFSSpec

Creates an `FSSpec` structure describing a file or directory. (Deprecated in Mac OS X v10.4. Use `FSMakeFSRefUnicode` (page 78) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr FSMakeFSSpec (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   FSSpec *spec
);
```

Parameters

vRefNum

A volume specification for the volume containing the file or directory. This parameter can contain a volume reference number, a drive number, or 0 to specify the default volume.

dirID

The parent directory ID of the target object. If the directory is sufficiently specified in the `fileName` parameter, the `dirID` parameter can be set to 0. If the `fileName` parameter contains an empty string, `FSMakeFSSpec` creates an `FSSpec` structure for the directory specified by the `dirID` parameter.

fileName

A full or partial pathname. If the `fileName` parameter specifies a full pathname, `FSMakeFSSpec` ignores both the `vRefNum` and `dirID` parameters. A partial pathname might identify only the final target, or it might include one or more parent directory names. If `fileName` specifies a partial pathname, then `vRefNum`, `dirID`, or both must be valid.

spec

A pointer to a file system specification to be filled in by `FSMakeFSSpec`. The `FSMakeFSSpec` function fills in the fields of the file system specification using the information contained in the other three parameters. If your application receives any result code other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0.

The file system specification structure that you pass in this parameter should not share storage space with the input pathname; the `name` field may be initialized to the empty string before the pathname has been processed. For example, `fileName` should not refer to the `name` field of the output file system specification.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

You should call `FSMakeFSSpec`, or one of the corresponding parameter block functions, `PBMakeFSSpecSync` (page 478) and `PBMakeFSSpecAsync` (page 477) , whenever you want to create an `FSSpec` structure. You should not create an `FSSpec` by filling in the fields of the structure yourself.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `FSMakeFSSpec` fills in the structure and then returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for other operations, such as creating a file with the `FSpCreate` (page 349) function.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the `vRefNum` parameter. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
MakeEffectMovie
Movie From DataRef
qtdataexchange
qtdataexchange.win
qtgraphics

Declared In
`Files.h`

FSpCatMove

Moves a file or directory from one location to another on the same volume. (Deprecated in Mac OS X v10.4. Use `FSMoveObject` (page 82) instead.)

```
OSErr FSpCatMove (
   const FSSpec *source,
   const FSSpec *dest
);
```

Parameters

source

A pointer to an `FSSpec` structure specifying the name and location of the file or directory to move. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

dest

A pointer to an `FSSpec` structure specifying the name and location of the directory into which the source file or directory is to be moved. The directory ID specified in the `parID` field of this `FSSpec` is the directory ID of the parent of the directory into which you want to move the source file or directory. The `name` field of this `FSSpec` specifies the name of the directory into which you want to move the source file or directory.

If you don't already know the parent directory ID of the destination directory, it might be easier to use the `PBCatMoveSync` or `PBCatMoveAsync` function, which allow you to specify only the directory ID of the destination directory.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSpCatMove` function is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. You cannot use `FSpCatMove` to move a file or directory to another volume (that is, the `vRefNum` field in both `FSSpec` structures in the `source` and `dest` parameters must be the same). Also, you cannot use `FSpCatMove` to rename files or directories; to rename a file or directory, use `FSpRename` (page 356).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 82) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 153) and `PBMoveObjectAsync` (page 152).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Not available to 64-bit applications.

Declared In
Files.h

FSpCreate

Creates a new file. (Deprecated in Mac OS X v10.4. Use FSCreateFileUnicode (page 54) instead.)

```
OSErr FSpCreate (
   const FSSpec *spec,
   OSType creator,
   OSType fileType,
   ScriptCode scriptTag
);
```

Parameters

spec

A pointer to an FSSpec structure specifying the file to be created. See FSSpec (page 226) for a description of the FSSpec data type.

creator

The creator of the new file. See the documentation for the Finder Interface for more information on file creators.

fileType

The file type of the new file. See the documentation for the Finder Interface for more information on file types.

scriptTag

The code of the script system in which the filename is to be displayed. If you have established the name and location of the new file using either the NavAskSaveChanges or NavCustomAskSaveChanges function, specify the script code returned in the reply structure. Otherwise, specify the system script by setting the scriptTag parameter to the value smSystemScript.

For more information about the functions NavAskSaveChanges and NavCustomAskSaveChanges, see *Programming With Navigation Services*. See the *Script Manager Reference* for a description of the smSystemScript constant.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The FSpCreate function creates a new file (both data and resource forks) with the specified type, creator, and script code. The new file is unlocked and empty. The date and time of creation and last modification are set to the current date and time.

Files created using FSpCreate are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, FSpOpenDF (page 354) , HOpenDF (page 368) , PBHOpenDFSync (page 460) or PBHOpenDFAsync (page 458).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager functions HCreateResFile or FSpCreateResFile to create a resource map in the file before you can open it by calling one of the Resource Manager functions HOpenResFile or FSpOpenResFile).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpCreate` is not available, you can use the function `HCreate` (page 363) instead. To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 54) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 125) and `PBCreateFileUnicodeAsync` (page 124).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
qtdataref
qtdataref.win
qtmoviefromprocs
ThreadsImporter
ThreadsImportMovie

Declared In
`Files.h`

FSpDelete

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `FSDeleteObject` (page 57) instead.)

```
OSErr FSpDelete (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to delete. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If you attempt to delete an open file or a non-empty directory, `FSpDelete` returns the result code `fBsyErr`. `FSpDelete` also returns the result code `fBsyErr` if the directory has an open working directory associated with it.

Discussion
If the specified target is a file, both forks of the file are deleted. The file ID reference, if any, is removed. A file must be closed before you can delete it. Similarly, a directory must be empty before you can delete it.

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpDelete` is not available, you can use the function `HDelete` (page 365) instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
MovieGWorlds
MovieToAIFF

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

qtdataexchange
qtdataref
qtdataref.win

Declared In
`Files.h`

FSpDirCreate

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `FSCreateDirectoryUnicode` (page 52) instead.)

```
OSErr FSpDirCreate (
   const FSSpec *spec,
   ScriptCode scriptTag,
   SInt32 *createdDirID
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the directory to be created.

Note that if the parent directory ID for the directory described by this `FSSpec` is 0 and the volume specified in this `FSSpec` is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the specified volume is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, set the parent directory ID to the constant `fsRtDirID`(2).

scriptTag

The code of the script system in which the directory name is to be displayed. If you have established the name and location of the new directory using either the `NavAskSaveChanges` or `NavCustomAskSaveChanges` function, specify the script code returned in the reply structure. Otherwise, specify the system script by setting the `scriptTag` parameter to the value `smSystemScript`.

For more information on the functions `NavAskSaveChanges` and `NavCustomAskSaveChanges`, see *Programming With Navigation Services*. For a description of the `smSystemScript` constant, see the *Script Manager Reference*.

createdDirID

On return, a pointer to the directory ID of the directory that was created.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSpDirCreate` function sets the date and time of creation and last modification to the current date and time.

Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpDirCreate` is not available, you can use the function `DirCreate` (page 345) instead. To create a directory with a Unicode name, use the function `FSCreateDirectoryUnicode` (page 52) , or one of the corresponding parameter block calls, `PBCreateDirectoryUnicodeSync` (page 123) and `PBCreateDirectoryUnicodeAsync` (page 122).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSpExchangeFiles

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `FSExchangeObjects` (page 59) instead.)

```
OSErr FSpExchangeFiles (
   const FSSpec *source,
   const FSSpec *dest
);
```

Parameters

source

A pointer to an `FSSpec` for the first file to swap. The contents of this file and its file information are placed in the file specified in the `dest` parameter. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

dest

A pointer to an `FSSpec` for the second file to swap. The contents of this file and its file information are placed in the file specified in the `source` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSpExchangeFiles` function swaps the data in two files by changing the information in the volume's catalog and, if either of the files are open, in the file control blocks. The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `FSpExchangeFiles` when updating an existing file, so that the file ID remains valid in case the file is being tracked through its file ID. The `FSpExchangeFiles` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

The FSpExchangeFiles function works on both open and closed files. If either file is open, FSpExchangeFiles updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access permissions as opening both files for writing.

The files whose data is to be exchanged must both reside on the same volume. If they do not, FSpExchangeFiles returns the result code diffVolErr.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the FSExchangeObjects (page 59) , PBExchangeObjectsSync (page 132) , or PBExchangeObjectsAsync (page 131) function.

Special Considerations

The "compatibility code," by which FSpExchangeFiles attempted to perform the file exchange itself if it suspected that the underlying filesystem did not have Exchange capability, has been removed in Mac OS 9 and X.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (fsRdWrPerm) to both files before calling FSpExchangeFiles. Exclusive read/write access to both files will ensure that FSpExchangeFiles doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

FSpExchangeFiles does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling FSpExchangeFiles ensures that the files are unlocked because locked files cannot be opened with write access.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

FSpGetFInfo

Obtains the Finder information for a file. (Deprecated in Mac OS X v10.4. Use FSGetCatalogInfo (page 67) instead.)

```
OSErr FSpGetFInfo (
   const FSSpec *spec,
   FInfo *fndrInfo
);
```

Parameters

spec

A pointer to an FSSpec structure specifying the file. See FSSpec (page 226) for a description of the FSSpec data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

fndrInfo

On return, a pointer to information used by the Finder. The `FSpGetFInfo` function returns the Finder information from the volume catalog entry for the specified file. The function provides only the original Finder information—the information in the `FInfo` or `DInfo` structures, not the information in the `FXInfo` or `DXInfo` structures. For a description of the `FInfo` structure, see the *Finder Interface Reference*.

Return Value
A result code. If the specified object is a folder, this function returns `fnfErr`. For other possible return values, see "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Movie From DataRef
qtdataexchange
qtdataexchange.win
qtgraphics.win
SoundPlayer.win

Declared In
`Files.h`

FSpOpenDF

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 87) instead.)

```
OSErr FSpOpenDF (
   const FSSpec *spec,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file whose data fork is to be opened. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

permission

A constant indicating the type of access with which to open the file's data fork. In most cases, you can simply set the `permission` parameter to `fsCurPerm`. Some applications request `fsRdWrPerm`, to ensure that they can both read from and write to a file. For a description of the types of access that you can request, see "File Access Permission Constants" (page 295).

refNum

On return, a pointer to the file reference number for accessing the open data fork.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpOpenDF` is not available, you can use the function `HOpenDF` (page 368) instead.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) and `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `FSpOpenDF` function, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
JPEG File Interchange Format
qtdataexchange
qtmoviefromprocs
qtmoviefromprocs.win
soundsnippets

Declared In
`Files.h`

FSpOpenRF

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 87) instead.)

```
OSErr FSpOpenRF (
   const FSSpec *spec,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

spec
A pointer to an `FSSpec` structure specifying the file whose resource fork is to be opened. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

permission
A constant indicating the type of access with which to open the file's resource fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295).

refNum
On return, a pointer to the file reference number for accessing the open resource fork.

Return Value
A result code. See "File Manager Result Codes" (page 329). On some file systems, `FSpOpenRF` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion
Before calling this function, you should call the `Gestalt` function to check that the function is available. If `FSpOpenRF` is not available, you can use the function `HOpenRF` (page 368) instead.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `FSpOpenRF` function, you will receive an error message.

Special Considerations

Generally, your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `FSpOpenRF` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

Movie From DataRef

qtcontroller

qtgraphics

qtgraphics.win

qtvectors

Declared In

`Files.h`

FSpRename

Renames a file or directory. (Deprecated in Mac OS X v10.4. Use `FSRenameUnicode` (page 100) instead.)

```
OSErr FSpRename (
   const FSSpec *spec,
   ConstStr255Param newName
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to rename. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

newName

The new name of the file or directory.

Return Value

A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
If a file ID reference for the specified file exists, it remains with the renamed file.

If you want to change the name of a new copy of an existing file, you should use the `FSpExchangeFiles` (page 352) function instead. To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 100) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 161) and `PBRenameUnicodeAsync` (page 161).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
MovieGWorlds
QT Internals

Declared In
`Files.h`

FSpRstFLock

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr FSpRstFLock (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file to unlock. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `FSpRstFLock` to unlock a directory. Otherwise, you can only use this function to unlock a file.

You can lock a file or directory with the `FSpSetFLock` (page 358) , `HSetFLock` (page 372) , `PBHSetFLockSync` (page 471) , and `PBHSetFLockAsync` (page 470) functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

FSpSetFInfo

Sets the Finder information about a file. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr FSpSetFInfo (
   const FSSpec *spec,
   const FInfo *fndrInfo
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file for which to set the Finder information. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

fndrInfo

A pointer to the new Finder information. For a description of the `FInfo` data type, see the *Finder Interface Reference*.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `FSpSetFInfo` function changes the Finder information in the volume catalog entry for the specified file. `FSpSetFInfo` allows you to set only the original Finder information—the information in the `FInfo` or `DInfo` structures, not the information in the `FXInfo` or `DXInfo` structures.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

FSCopyObject

Declared In

`Files.h`

FSpSetFLock

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr FSpSetFLock (
   const FSSpec *spec
);
```

Parameters

spec

A pointer to an `FSSpec` structure specifying the file or directory to lock. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) functions indicate that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `FSpSetFLock` to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. This function has no effect on existing access paths.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

FSRead

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.4. Use `FSReadFork` (page 98) instead.)

```
OSErr FSRead (
    FSIORefNum refNum,
    SInt32 *count,
    void *buffPtr
);
```

Parameters

refNum
The file reference number of the open file from which to read.

count
On input, a pointer to the number of bytes to read; on output, a pointer to the number of bytes actually read.

buffPtr
A pointer to the data buffer into which the data will be read. This buffer is allocated by your application and must be at least as large as the `count` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Because the read operation begins at the current mark, you might want to set the mark first by calling the `SetFPos` (page 500) function. If you try to read past the logical end-of-file, `FSRead` reads in all the data up to the end-of-file, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `FSRead` moves the file mark to the byte following the last byte read and returns `noErr`.

The low-level functions `PBReadSync` and `PBReadAsync` let you set the mark without having to call `SetFPos`. Furthermore, if you want to read data in newline mode, you must use `PBReadSync` or `PBReadAsync` instead of `FSRead`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

If you wish to read from named forks other than the data or resource forks, or from files larger than 2GB, you must use the `FSReadFork` (page 98) function, or one of its corresponding parameter block calls, `PBReadForkSync` (page 159) and `PBReadForkAsync` (page 158). If you attempt to use `FSRead` to read from a file larger than 2GB, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Inside Mac ICM Code
JPEG File Interchange Format
qtmoviefromprocs
qtmoviefromprocs.win
ThreadsImportMovie

Declared In
`Files.h`

FSWrite

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.4. Use `FSWriteFork` (page 106) instead.)

```
OSErr FSWrite (
    FSIORefNum refNum,
    SInt32 *count,
    const void *buffPtr
);
```

Parameters

refNum
The file reference number of the open file to which to write.

count
On input, a pointer to the number of bytes to write to the file. Passing 0 in this parameter will return a `paramErr` error.

On output, a pointer to the number of bytes actually written.

buffPtr
A pointer to the data buffer containing the data to write.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `FSWrite` function takes the specified number of bytes from the data buffer and attempts to write them to the file. Because the write operation begins at the current mark, you might want to set the mark first by calling the `SetFPos` (page 500) function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

If the write operation completes successfully, `FSWrite` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `FSWrite` moves the logical end-of-file. If you try to write past the physical end-of-file, `FSWrite` adds one or more clumps to the file and moves the physical end-of-file accordingly.

The low-level functions `PBWriteSync` and `PBWriteAsync` let you set the mark without having to call `SetFPos`.

If you wish to write to named forks other than the data or resource forks, or grow files larger than 2GB, you must use the `FSWriteFork` (page 106) function, or one of its corresponding parameter block calls, `PBWriteForkSync` (page 171) and `PBWriteForkAsync` (page 170). If you attempt to use `FSWrite` to write to a file larger than 2GB, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
JPEG File Interchange Format
qtinfo
qtinfo.win
soundsnippets
soundsnippets.win

Declared In
`Files.h`

GetEOF

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use `FSGetForkSize` (page 73) instead.)

```
OSErr GetEOF (
   FSIORefNum refNum,
   SInt32 *logEOF
);
```

Parameters

`refNum`
The file reference number of an open file.

`logEOF`
On return, a pointer to the logical size (the logical end-of-file) of the given file.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the `FSGetForkSize` (page 73) function, or one of the corresponding parameter block functions, `PBGetForkSizeSync` (page 145) and `PBGetForkSizeAsync` (page 145).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
JPEG File Interchange Format
qtmoviefromprocs
qtmoviefromprocs.win
ThreadsImporter
ThreadsImportMovie

Declared In
`Files.h`

GetFPos

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use `FSGetForkPosition` (page 72) instead.)

```
OSErr GetFPos (
   FSIORefNum refNum,
   SInt32 *filePos
);
```

Parameters

`refNum`
The file reference number of an open file.

`filePos`
On return, a pointer to the current position of the mark. The position value is zero-based; that is, the value of `filePos` is 0 if the file mark is positioned at the beginning of the file.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Because the read and write operations performed by the functions `FSRead` (page 359) and `FSWrite` (page 360) begin at the current mark, you should call `GetFPos`, or one of the parameter block functions, `PBGetFPosSync` (page 436) and `PBGetFPosAsync` (page 435) , to determine the current position of the file mark before reading from or writing to the file.

To determine the current position of a named fork, or of a fork larger than 2GB, use the `FSGetForkPosition` (page 72) function, or one of the corresponding parameter block calls, `PBGetForkPositionSync` (page 144) and `PBGetForkPositionAsync` (page 143).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Inside Mac ICM Code

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

GetVRefNum

Gets a volume reference number from a file reference number. (Deprecated in Mac OS X v10.4. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr GetVRefNum (
   FSIORefNum fileRefNum,
   FSVolumeRefNum *vRefNum
);
```

Parameters

`fileRefNum`
The file reference number of an open file.

`vRefNum`
On return, a pointer to the volume reference number of the volume containing the file specified in the `refNum` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If you also want to determine the directory ID of the specified file's parent directory, call the `PBGetFCBInfoSync` (page 432) or `PBGetFCBInfoAsync` (page 431) functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
qtdataexchange
qtmoviefromprocs
qtmoviefromprocs.win
qtshortcut
qtshortcut.win

Declared In
`Files.h`

HCreate

Creates a new file. (Deprecated in Mac OS X v10.4. Use `FSCreateFileUnicode` (page 54) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr HCreate (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   OSType creator,
   OSType fileType
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID of the directory in which to create the new file.

fileName

The name of the new file. This can be a full or partial pathname.

You should not allow users to give files names that begin with a period (.). This ensures that files can be successfully opened by applications calling `HOpen` (page 367) instead of `HOpenDF` (page 368).

creator

The creator of the new file. For more information on a file's creator, see the Finder Interface documentation.

fileType

The file type of the new file. For more information on a file's type, see the Finder Interface documentation.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `HCreate` function creates a new file (both data and resource forks) with the specified name, creator, and file type. The new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time.

Files created using `HCreate` are not automatically opened. If you want to write data to the new file, you must first open the file using a file access function.

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager functions `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 54) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 125) and `PBCreateFileUnicodeAsync` (page 124).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

HDelete

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `FSDeleteObject` (page 57) instead.)

```
OSErr HDelete (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID of the parent directory of the file or directory to delete.

fileName

The name of the file or directory to delete. This can be a full or partial pathname.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you attempt to delete an open file or a non-empty directory, `HDelete` returns the result code `fBsyErr`. `HDelete` also returns the result code `fBsyErr` if the directory has an open working directory associated with it.

Discussion

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that reference is removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HGetFInfo

Obtains the Finder information for a file. (Deprecated in Mac OS X v10.4. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr HGetFInfo (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   FInfo *fndrInfo
);
```

Parameters

vRefNum

The volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

fileName
The name of the file.

fndrInfo
On return, a pointer to the Finder information stored in the specified volume's catalog. The function returns only the original Finder information—that contained in an `FInfo` structure, not that in an `FXInfo` structure.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HGetVol

Determines the current default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr HGetVol (
   StringPtr volName,
   FSVolumeRefNum *vRefNum,
   SInt32 *dirID
);
```

Parameters
volName
On return, a pointer to the name of the default volume. If you do not want the name of the default volume returned, set this parameter to `NULL`.

vRefNum
On return, a pointer to the volume reference number of the default volume.

dirID
On return, a pointer to the directory ID of the default directory.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Version Notes
When CarbonLib is not present, the `HGetVol` function returns a working directory reference number in the `vRefNum` parameter if the previous call to `HSetVol` (page 373) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Carbon Porting Notes

Carbon applications should use `HGetVol` and `HSetVol` to get and set the default directory. the functions `GetVol` and `SetVol`, as well as working directories, are no longer supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Not available to 64-bit applications.

Related Sample Code
MakeEffectMovie
Movie From DataRef
qtcontroller
qtgraphics
qtvectors

Declared In
`Files.h`

HOpen

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 87) instead.)

```
OSErr HOpen (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum
A volume reference number, drive number, or 0 for the default volume.

dirID
The directory ID of the file's parent directory.

fileName
The name of the file.

permission
A constant specifying the type of access with which to open the file's data fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295).

refNum
On return, a pointer to the file reference number for accessing the open fork.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If you use `HOpen` to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use `HOpenDF` instead of `HOpen`.

Special Considerations

If you use `HOpen` to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use `HOpenDF` (page 368) instead of `HOpen`.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HOpenDF

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 87) instead.)

```
OSErr HOpenDF (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum
A volume reference number, drive number, or 0 for the default volume.

dirID
The directory ID of the file's parent directory.

fileName
The name of the file.

permission
A constant specifying the type of access with which to open the file's data fork. For a description of the types of access which you can request, see "File Access Permission Constants" (page 295).

refNum
On return, a pointer to the file reference number for accessing the open data fork.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of the corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `HOpenDF` function, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HOpenRF

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `FSOpenFork` (page 87) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr HOpenRF (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   SInt8 permission,
   FSIORefNum *refNum
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The directory ID for the file's parent directory.

fileName

The name of the file.

permission

A constant specifying the type of access with which to open the file's resource fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295).

refNum

On return, a pointer to the file reference number for accessing the open resource fork.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you try to open the resource fork of a file for which no resource fork exists with read-only access, `HOpenRF` returns the error `eofErr`.

Discussion

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `HOpenRF` function, you will receive an error message.

Special Considerations

Generally, your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `HOpenRF` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HRename

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `FSRenameUnicode` (page 100) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr HRename (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param oldName,
   ConstStr255Param newName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

A directory ID.

oldName

An existing filename, directory name, or volume name.

newName

The new filename, directory name, or volume name.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

Given the name of a file or directory in the `oldName` parameter, `HRename` changes it to the name in the `newName` parameter. Given a volume name in the `oldName` parameter or a volume reference number in the `vRefNum` parameter, `HRename` changes the name of the volume to the name in `newName`. Access paths currently in use aren't affected by this function.

If a file ID reference exists for a file you are renaming, the file ID remains with the renamed file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 100) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 161) and `PBRenameUnicodeAsync` (page 161).

Special Considerations

You cannot use `HRename` to change the directory in which a file resides. To move a file or directory, use the `FSpCatMove` (page 348), `PBCatMoveSync` (page 380), or `PBCatMoveAsync` (page 379) functions.

If you're renaming a volume, make sure that both names end with a colon.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HRstFLock

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr HRstFLock (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to unlock.

fileName

The name of the file or directory.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `HRstFLock` to unlock a directory. Otherwise, you can only use this function to unlock a file.

You can lock a file or directory with the `FSpSetFLock` (page 358) , `HSetFLock` (page 372) , `PBHSetFLockSync` (page 471) , and `PBHSetFLockAsync` (page 470) functions.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

HSetFInfo

Sets the Finder information for a file. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr HSetFInfo (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName,
   const FInfo *fndrInfo
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file.

fileName

The name of the file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

fndrInfo

A pointer to the new Finder information. The function changes the Finder information stored in the volume's catalog. `HSetFInfo` changes only the original Finder information—that contained in an `FInfo` structure, not that contained in an `FXInfo` structure. For a description of the `FInfo` data type, see the *Finder Interface Reference*.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

HSetFLock

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr HSetFLock (
   FSVolumeRefNum vRefNum,
   SInt32 dirID,
   ConstStr255Param fileName
);
```

Parameters

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

The parent directory ID of the file or directory to lock.

fileName

The name of the file or directory.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `HSetFLock` to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. This function has no effect on existing access paths.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

HSetVol

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr HSetVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum,
   SInt32 dirID
);
```

Parameters

volName

The name of a mounted volume or the partial pathname of a directory. This parameter can be `NULL`.

vRefNum

A volume reference number, drive number, or 0 for the default volume.

dirID

A directory ID.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `HSetVol` function lets you specify the default directory by volume reference number or by directory ID.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 0.

Special Considerations

Use of the `HSetVol` function is discouraged if your application may execute in system software versions prior to version 7.0. Because the specified directory might not itself be a working directory, `HSetVol` records the default volume and directory separately, using the volume reference number of the volume and the actual directory ID of the specified directory. Subsequent calls to `GetVol` (or `PBGetVolSync` or `PBGetVolAsync`) return only the volume reference number, which will cause that volume's root directory (rather than the default directory, as expected) to be accessed.

Carbon Porting Notes

Carbon applications should use `HGetVol` and `HSetVol` to get and set the default directory. the functions `GetVol` and `SetVol`, as well as working directories, are no longer supported.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBAllocateAsync

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkAsync` (page 112) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBAllocateAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 248) variant of the basic File Manager parameter block. See ParamBlockRec (page 252) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ioRefNum

On input, a file reference number for the file to which to allocate additional blocks.

ioReqCount

On input, the number of bytes to allocate.

ioActCount

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The PBAllocateAsync function adds ioReqCount bytes to the specified file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, PBAllocateAsync allocates the rest of the space on the volume and returns dskFulErr as its function result.

If the total number of requested bytes is unavailable, PBAllocateAsync allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call PBAllocContigAsync (page 376) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the PBAllocateAsync function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 499) function, or by writing data to the file with the FSWrite (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 499) function, or one of the related parameter block calls, PBSetEOFSync (page 484) and PBSetEOFAsync (page 483).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the `PBAllocateAsync` function does not return the number of bytes actually allocated in the `ioActCount` field.

To determine the remaining space on a volume before calling `PBAllocateAsync`, use the functions `PBXGetVolInfoSync` or `PBXGetVolInfoAsync`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBAllocateSync

Allocates additional space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkSync` (page 113) instead.)

```
OSErr PBAllocateSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for the file to which to allocate additional blocks.

`ioReqCount`

On input, the number of bytes to allocate.

`ioActCount`

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The `PBAllocateSync` function adds `ioReqCount` bytes to the specified file and sets the physical end-of-file to 1 byte beyond the last block allocated. If there isn't enough empty space on the volume to satisfy the allocation request, `PBAllocateSync` allocates the rest of the space on the volume and returns `dskFulErr` as its function result.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

If the total number of requested bytes is unavailable, `PBAllocateSync` allocates whatever space, contiguous or not, is available. To force the allocation of the entire requested space as a contiguous piece, call `PBAllocContigSync` (page 378) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `PBAllocateSync` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 499) function, or by writing data to the file with the `FSWrite` (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 499) function, or one of the related parameter block calls, `PBSetEOFSync` (page 484) and `PBSetEOFAsync` (page 483).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, if there is not enough space left on the volume to allocate the requested number of bytes, the `PBAllocateSync` function does not return the number of bytes actually allocated in the `ioActCount` field.

To determine the remaining space on a volume before calling `PBAllocateSync`, use the functions `PBXGetVolInfoSync` or `PBXGetVolInfoAsync`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBAllocContigAsync

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkAsync` (page 112) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBAllocContigAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 248) variant of the basic File Manager parameter block. See ParamBlockRec (page 252) for a description of the ParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ioRefNum

On input, a file reference number for the open file.

ioReqCount

On input, the number of bytes to allocate.

ioActCount

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The PBAllocContigAsync function is identical to the PBAllocateAsync (page 373) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, PBAllocContigAsync does nothing and returns dskFulErr as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call PBAllocateAsync (page 373) instead.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the PBAllocContigAsync function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the SetEOF (page 499) function, or by writing data to the file with the FSWrite (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the SetEOF (page 499) function, or one of the related parameter block calls, PBSetEOFSync (page 484) and PBSetEOFAsync (page 483).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, `PBAllocContigAsync` does not return 0 in the `ioActCount` field, so your application cannot rely upon this value.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBAllocContigSync

Allocates additional contiguous space on a volume to an open file. (Deprecated in Mac OS X v10.4. Use `PBAllocateForkSync` (page 113) instead.)

```
OSErr PBAllocContigSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for the open file.

`ioReqCount`

On input, the number of bytes to allocate.

`ioActCount`

On output, the number of bytes actually allocated, rounded up to the nearest multiple of the allocation block size.

The `PBAllocContigSync` function is identical to the `PBAllocateSync` (page 375) function except that if there isn't enough contiguous empty space on the volume to satisfy the allocation request, `PBAllocContigSync` does nothing and returns `dskFulErr` as its function result. If you want to allocate whatever space is available, even when the entire request cannot be filled by the allocation of a contiguous piece, call `PBAllocateSync` (page 375) instead.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

The File Manager automatically allocates file blocks if you move the logical end-of-file past the physical end-of-file, and it automatically deallocates unneeded blocks from a file if you move the logical end-of-file to a position more than one allocation block before the current physical end-of-file. Consequently, you do not in general need to be concerned with allocating or deallocating file blocks. However, you can improve file block contiguity if you use the `PBAllocContigSync` function to preallocate file blocks. This is most useful if you know in advance how big a file is likely to become.

The space allocated with this function is not permanently assigned to the file until the file's logical end-of-file is changed to include the allocated space. When a file (or volume) is closed, the space beyond the file's logical EOF is made available for other purposes, even if previously allocated to the file with a call to this function. You can change the end-of-file by setting it with the `SetEOF` (page 499) function, or by writing data to the file with the `FSWrite` (page 360) function.

This function is not supported by all file systems; for example, volumes mounted by the AppleShare file system do not support this function. To allocate space for a file on any volume, use the `SetEOF` (page 499) function, or one of the related parameter block calls, `PBSetEOFSync` (page 484) and `PBSetEOFAsync` (page 483).

To allocate space for a file beyond 2 GB, use the `FSAllocateFork` (page 43) function, or one of the corresponding parameter block functions, `PBAllocateForkSync` (page 113) and `PBAllocateForkAsync` (page 112).

Special Considerations

In Mac OS 7.5.5 through Mac OS 8.5, when there is not enough space to allocate the requested number of bytes, `PBAllocContigSync` does not return 0 in the `ioActCount` field, so your application cannot rely upon this value.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBCatMoveAsync

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use `PBMoveObjectAsync` (page 152) instead.)

```
OSErr PBCatMoveAsync (
   CMovePBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog move parameter block. See `CMovePBRec` (page 188) for a description of the `CMovePBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). This function returns `permErr` if called on a locked file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to the name of the file or directory to move.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioNewName`
: On input, a pointer to the name of the destination directory. Pass `NULL` in this field if you wish to specify the destination directory by its directory ID.

`ioNewDirID`
: On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, this is the parent directory ID of the directory into which the file or directory is to be moved. It is usually simplest to specify the destination directory by passing its directory ID in the `ioNewDirID` field and by setting `ioNewName` to `NULL`.

`ioDirID`
: On input, the parent directory ID of the file or directory to move.

`PBCatMoveAsync` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. If a file ID reference exists for the file, the file ID reference remains with the moved file.

The `PBCatMoveAsync` function cannot move a file or directory to another volume (that is, the value in the `ioVRefNum` field is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `FSpRename` (page 356) , `PBHRenameSync` (page 466) , or `PBHRenameAsync` (page 465) .

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 82) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 153) and `PBMoveObjectAsync` (page 152).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBCatMoveSync

Moves files or directories from one directory to another on the same volume. (Deprecated in Mac OS X v10.4. Use `PBMoveObjectSync` (page 153) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBCatMoveSync (
   CMovePBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a catalog move parameter block. See `CMovePBRec` (page 188) for a description of the `CMovePBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). This function returns `permErr` if called on a locked file.

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the file or directory to move.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioNewName`

On input, a pointer to the name of the destination directory. Pass `NULL` in this field if you wish to specify the destination directory by its directory ID.

`ioNewDirID`

On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, this is the parent directory ID of the destination directory. It is usually simplest to specify the destination directory by passing its directory ID in the `ioNewDirID` field and by setting `ioNewName` to `NULL`.

`ioDirID`

On input, the parent directory ID of the file or directory to move.

`PBCatMoveSync` is strictly a file catalog operation; it does not actually change the location of the file or directory on the disk. If a file ID reference exists for the file, the file ID reference remains with the moved file.

The `PBCatMoveSync` function cannot move a file or directory to another volume (that is, the value in the `ioVRefNum` field is used in specifying both the source and the destination). Also, you cannot use it to rename files or directories; to rename a file or directory, use `FSpRename` (page 356) , `PBHRenameSync` (page 466) , or `PBHRenameAsync` (page 465).

If you need to move files or directories with named forks other than the data and resource forks, with long Unicode names, or files larger than 2GB, you should use the `FSMoveObject` (page 82) function, or one of the corresponding parameter block calls, `PBMoveObjectSync` (page 153) and `PBMoveObjectAsync` (page 152).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBCatSearchAsync

Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated in Mac OS X v10.4. Use `PBCatalogSearchAsync` (page 114) instead.)

```
OSErr PBCatSearchAsync (
   CSParamPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CSParam` (page 193) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. When `PBCatSearchAsync` has searched the entire volume, it returns `eofErr`. If it exits because it either spends the maximum time allowed in the `ioSearchTime` field or finds the maximum number of matches allowed in the `ioReqMatchCount` field, it returns `noErr`.

`ioNamePtr`

On input, a pointer to the name of the volume to search.

`ioVRefNum`

On input, a volume reference number or drive number for the volume to search; or 0 for the default volume.

`ioMatchPtr`

On input, a pointer to an array of `FSSpec` (page 226) structure to hold the matches found by this function. On return, the `FSSpec` structures in this array identify the files and directories that match the criteria.

`ioReqMatchCount`

On input, the maximum number of matches to return.

`ioActMatchCount`

On output, the actual number of matches returned.

`ioSearchBits`

On input, a bitmap specifying the fields in the criteria structures to match against. See "Catalog Search Masks" (page 286) for a description of the bits in this field.

`ioSearchInfo1`

On input, a pointer to a `CInfoPBRec` (page 187) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in `ioSearchInfo2`, and set the matching value in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioSearchInfo2`
: On input, a pointer to a `CInfoPBRec` (page 187) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the matching value in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

`ioSearchTime`
: On input, the maximum allowed search time. If you pass 0 in this field, no time limit is set.

`ioCatPosition`
: The current catalog position, specified as a `CatPositionRec` (page 187) structure. You can use this field, along with the `ioSearchTime` field, to search a volume in segments. To search a volume in segments, set a time limit for the search in the `ioSearchTime` field and set the `initialize` field of the `CatPositionRec` structure to the location for the start of the search (0 if you wish to start searching at the beginning of the volume). On return, the catalog position will be updated. You can then pass this updated `CatPositionRec` structure to the next call to `PBCatSearchSync` to continue searching at the place where you left off.

`ioOptBuffer`
: On input, a pointer to an optional read buffer.

`ioOptBufSize`
: On input, the length of the optional read buffer.

If the catalog file changes between two timed calls to `PBCatSearchAsync` (that is, when you are using `ioSearchTime` and `ioCatPosition` to search a volume in segments and the catalog file changes between searches), `PBCatSearchAsync` returns a result code of `catChangedErr` and no matches. Depending on what has changed on the volume, `ioCatPosition` might be invalid, most likely by a few entries in one direction or another. You can continue the search, but you risk either skipping some entries or reading some twice.

Special Considerations

Not all volumes support the `PBCatSearchAsync` function. Before you call `PBCatSearchAsync` to search a particular volume, you should call the `PBHGetVolParmsAsync` (page 517) function to determine whether that volume supports `PBCatSearchAsync`. If the `bHasCatSearch` bit is set in the `vMAttrib` field, then the volume supports `PBCatSearchAsync`.

Even though AFP volumes support `PBCatSearchSync`, they do not support all of its features that are available on local volumes. These restrictions apply to AFP volumes:

- AFP volumes do not use the `ioSearchTime` field. Current versions of the AppleShare server software search for 1 second or until 4 matches are found. The AppleShare workstation software keeps requesting the appropriate number of matches until the server returns either the number specified in the `ioReqMatchCount` field or an error.
- AFP volumes do not support both logical and physical fork lengths. If you request a search using the length of a fork, the actual minimum length used is the smallest of the values in the logical and physical fields of the `ioSearchInfo1` structure and the actual maximum length used is the largest of the values in the logical and physical fields of the `ioSearchInfo2` structure.
- The `fsSBNegate` bit of the `ioSearchBits` field is ignored during searches of remote volumes that support AFP version 2.1.
- If the AFP server returns `afpCatalogChanged`, the catalog position structure returned to your application (in the `ioCatPosition` field) is the same one you passed to `PBCatSearchAsync`. You should clear the `initialize` field of that structure to restart the search from the beginning.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBCatSearchSync

Searches a volume's catalog file using a set of search criteria that you specify. (Deprecated in Mac OS X v10.4. Use `PBCatalogSearchSync` (page 116) instead.)

```
OSErr PBCatSearchSync (
   CSParamPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CSParam` (page 193) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). When `PBCatSearchSync` has searched the entire volume, it returns `eofErr`. If it exits because it either spends the maximum time allowed in the `ioSearchTime` field or finds the maximum number of matches allowed in the `ioReqMatchCount` field, it returns `noErr`.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the volume to search.

`ioVRefNum`
On input, a volume reference number or drive number for the volume to search; or 0 for the default volume.

`ioMatchPtr`
On input, a pointer to an array of `FSSpec` (page 226) structure to hold the matches found by this function. On return, the `FSSpec` structures in this array identify the files and directories that match the criteria.

`ioReqMatchCount`
On input, the maximum number of matches to return.

`ioActMatchCount`
On output, the actual number of matches returned.

`ioSearchBits`
On input, a bitmap specifying the fields in the criteria structures to match against. See "Catalog Search Masks" (page 286) for a description of the bits in this field.

`ioSearchInfo1`
On input, a pointer to a `CInfoPBRec` (page 187) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in the structure passed in `ioSearchInfo2`, and set the matching value in this structure. For values that match against a range (such as dates), set the lower bounds for the range in this structure.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioSearchInfo2`
: On input, a pointer to a `CInfoPBRec` (page 187) union containing search information. For values that match by mask and value (Finder information, for example), set the bits in this structure, and set the matching value in the structure passed in the `ioSearchInfo1` field. For values that match against a range (such as dates), set the upper bounds for the range in this structure.

`ioSearchTime`
: On input, the maximum allowed search time. If you pass 0 in this field, no time limit is set.

`ioCatPosition`
: The current catalog position, specified as a `CatPositionRec` (page 187) structure. You can use this field, along with the `ioSearchTime` field, to search a volume in segments. To search a volume in segments, set a time limit for the search in the `ioSearchTime` field and set the `initialize` field of the `CatPositionRec` structure to the location for the start of the search (0 if you wish to start searching at the beginning of the volume). On return, the catalog position will be updated. You can then pass this updated `CatPositionRec` structure to the next call to `PBCatSearchSync` to continue searching at the place where you left off.

`ioOptBuffer`
: On input, a pointer to an optional read buffer.

`ioOptBufSize`
: On input, the length of the optional read buffer.

If the catalog file changes between two timed calls to `PBCatSearchSync` (that is, when you are using `ioSearchTime` and `ioCatPosition` to search a volume in segments and the catalog file changes between searches), `PBCatSearchSync` returns a result code of `catChangedErr` and no matches. Depending on what has changed on the volume, `ioCatPosition` might be invalid, most likely by a few entries in one direction or another. You can continue the search, but you risk either skipping some entries or reading some twice.

Special Considerations

Not all volumes support the `PBCatSearchSync` function. Before you call `PBCatSearchSync` to search a particular volume, you should call the `PBHGetVolParmsSync` (page 518) function to determine whether that volume supports `PBCatSearchSync`.If the `bHasCatSearch` bit is set in the `vMAttrib` field, then the volume supports `PBCatSearchSync`.

Even though AFP volumes support `PBCatSearchSync`, they do not support all of its features that are available on local volumes. These restrictions apply to AFP volumes:

- AFP volumes do not use the `ioSearchTime` field. Current versions of the AppleShare server software search for 1 second or until 4 matches are found. The AppleShare workstation software keeps requesting the appropriate number of matches until the server returns either the number specified in the `ioReqMatchCount` field or an error.
- AFP volumes do not support both logical and physical fork lengths. If you request a search using the length of a fork, the actual minimum length used is the smallest of the values in the logical and physical fields of the `ioSearchInfo1` structure and the actual maximum length used is the largest of the values in the logical and physical fields of the `ioSearchInfo2` structure.
- The `fsSBNegate` bit of the `ioSearchBits` field is ignored during searches of remote volumes that support AFP version 2.1.
- If the AFP server returns `afpCatalogChanged`, the catalog position structure returned to your application (in the `ioCatPosition` field) is the same one you passed to `PBCatSearchSync`. You should clear the `initialize` field of that structure to restart the search from the beginning.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDirCreateAsync

Creates a new directory. (Deprecated in Mac OS X v10.4. Use `PBCreateDirectoryUnicodeAsync` (page 122) instead.)

```
OSErr PBDirCreateAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name for the new directory.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the `ioVRefNum` field is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant `fsRtDirID` (2) in this field. On output, the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

The `PBDirCreateAsync` function is identical to `PBHCreateAsync` (page 438) except that it creates a new directory instead of a file. The date and time of the directory's creation and last modification are set to the current date and time.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

To create a directory with a Unicode name, use the function FSCreateDirectoryUnicode (page 52) , or one of the corresponding parameter block calls, PBCreateDirectoryUnicodeSync (page 123) and PBCreateDirectoryUnicodeAsync (page 122).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBDirCreateSync

Creates a new directory. (Deprecated in Mac OS X v10.4. Use PBCreateDirectoryUnicodeSync (page 123) instead.)

```
OSErr PBDirCreateSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the HFileParam (page 238) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to the name for the new directory.

ioVRefNum
On input, a volume reference number, drive number, or 0 for the default volume.

ioDirID
On input, the parent directory ID. If the parent directory ID is 0 and the volume specified in the ioVRefNum field is the default volume, the new directory is placed in the default directory of the volume. If the parent directory ID is 0 and the volume specified in the ioVRefNum field is a volume other than the default volume, the new directory is placed in the root directory of the volume. To create a directory at the root of a volume, regardless of whether that volume is the current default volume, pass the constant fsRtDirID (2) in this field. On output, the directory ID of the new directory. Note that a directory ID, unlike a volume reference number, is a long integer.

The PBDirCreateSync function is identical to PBHCreateSync (page 439) except that it creates a new directory instead of a file. The date and time of the directory's creation and last modification are set to the current date and time.

To create a directory with a Unicode name, use the function FSCreateDirectoryUnicode (page 52) , or one of the corresponding parameter block calls, PBCreateDirectoryUnicodeSync (page 123) and PBCreateDirectoryUnicodeAsync (page 122).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddAPPLAsync

Adds an application to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to which you wish to add an application.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDirID`
On input, the ID of the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddAPPLSync

Adds an application to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to which you wish to add an application.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDirID`
On input, the ID of the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBDTAddIconAsync

Adds an icon definition to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddIconAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`

On input, the desktop database reference number of the database to which you wish to add an icon.

`ioTagInfo`

Reserved; on input, this field must be set to 0.

`ioDTBuffer`

On input, a pointer to the buffer holding the icon's bitmap.

`ioDTReqCount`

On input, the size in bytes of the buffer that you've allocated for the icon's bitmap. This value depends on the icon type. Be sure to allocate enough storage for the icon data 1024 bytes is the largest amount required for any icon in System 7. For a description of the values you can use to indicate the icon's size, see "Icon Size Constants" (page 307).

`ioIconType`

On input, the icon type. See "Icon Type Constants" (page 308) for a description of the values you can use in this field.

`ioFileCreator`

On input, the icon's file creator.

`ioFileType`

On input, the icon's file type.

If the database already contains an icon definition for an icon of that type, file type, and file creator, the new definition replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTAddIconSync

Adds an icon definition to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTAddIconSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number of the database to which you wish to add an icon.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTBuffer`
On input, a pointer to the buffer holding the icon's bitmap.

`ioDTReqCount`
On input, the size in bytes of the buffer that you've allocated for the icon's bitmap. This value depends on the icon type. Be sure to allocate enough storage for the icon data 1024 bytes is the largest amount required for any icon in System 7 For a description of the values you can use to indicate the icon's size, see "Icon Size Constants" (page 307).

`ioIconType`
On input, the icon type. See "Icon Type Constants" (page 308) for a description of the values you can use in this field.

`ioFileCreator`
On input, the icon's file creator.

`ioFileType`
On input, the icon's file type.

If the database already contains an icon definition for an icon of that type, file type, and file creator, the new definition replaces the old.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTCloseDown

Closes the desktop database, though your application should never do this itself. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTCloseDown (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant field of the parameter block for this function is:

`ioDTRefNum`

On input, the desktop database reference number.

System software uses the `PBDTCloseDown` function to close the desktop database; your application should never use this function, which is described here only for completeness. The system software closes the database when the volume is unmounted.

`PBDTCloseDown` runs synchronously only, and though it will not close down the desktop databases of remote volumes, it will invalidate all local desktop database reference values for remote desktop databases.

When the `PBDTCloseDown` function closes the database, it frees all resources allocated by `PBDTOpenInform` (page 409) or `PBDTGetPath` (page 408).

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBDTDeleteAsync

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTDeleteAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See DTPBRec (page 199) for a description of the DTPBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The PBDTDeleteAsync function removes the desktop database from a local volume. You can call PBDTDeleteAsync only when the database is closed. Your application should not call PBDTDeleteAsync unless absolutely necessary.

The relevant fields of the parameter block for this function are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function. See "File Manager Result Codes".

ioVRefNum

On input, the volume reference number of the desktop database to remove.

ioIndex

Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBDTDeleteSync

Removes the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTDeleteSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBDTDeleteSync` function removes the desktop database from a local volume. You can call `PBDTDeleteSync` only when the database is closed. Your application should not call `PBDTDeleteSync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioVRefNum`

On input, the volume reference number of the desktop database to remove.

`ioIndex`

Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTFlushAsync

Saves your changes to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTFlushAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
If your application adds information to or removes information from the desktop database, use the `PBDTFlushAsync` function to save your changes. The `PBDTFlushAsync` function writes the contents of the desktop database specified in the `ioDTRefNum` field to the volume.

The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to flush.

You must call `PBDTFlushAsync` or `PBDTFlushSync` (page 396) to update the copy of the desktop database stored on the volume if your application has manipulated information in the database using any of the following functions:

- `PBDTAddIconSync` (page 391)
- `PBDTAddIconAsync` (page 390)
- `PBDTAddAPPLSync` (page 389)
- `PBDTAddAPPLAsync` (page 388)
- `PBDTSetCommentSync` (page 416)
- `PBDTSetCommentAsync` (page 415)
- `PBDTRemoveAPPLSync` (page 411)
- `PBDTRemoveAPPLAsync` (page 410)
- `PBDTRemoveCommentSync` (page 412)
- `PBDTRemoveCommentAsync` (page 411)

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBDTFlushSync

Saves your changes to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTFlushSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See DTPBRec (page 199) for a description of the DTPBRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If your application adds information to or removes information from the desktop database, use the PBDTFlushSync function to save your changes. The PBDTFlushSync function writes the contents of the desktop database specified in the ioDTRefNum field to the volume.

The relevant field of the parameter block for this function is:

ioDTRefNum

On input, the desktop database reference number of the desktop database to flush.

You must call PBDTFlushSync or PBDTFlushAsync (page 394) to update the copy of the desktop database stored on the volume if your application has manipulated information in the database using any of the following functions:

- PBDTAddIconSync (page 391)
- PBDTAddIconAsync (page 390)
- PBDTAddAPPLSync (page 389)
- PBDTAddAPPLAsync (page 388)
- PBDTSetCommentSync (page 416)
- PBDTSetCommentAsync (page 415)
- PBDTRemoveAPPLSync (page 411)
- PBDTRemoveAPPLAsync (page 410)
- PBDTRemoveCommentSync (page 412)
- PBDTRemoveCommentAsync (page 411)

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBDTGetAPPLAsync

Identifies the application that can open a file with a given creator. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See DTPBRec (page 199) for a description of the DTPBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code. See "File Manager Result Codes".

ioNamePtr

On output, a pointer to the application's name.

ioDTRefNum

On input, the desktop database reference number of the desktop database containing the specified application.

ioIndex

On input, an index into the application list.

ioTagInfo

On output, the application's creation date.

ioFileCreator

On input, the signature of the application.

ioAPPLParID

On output, the application's parent directory.

A single call, with the ioIndex field set to 0, finds the application file with the most recent creation date. If you want to retrieve information about all copies of the application with the given signature, start with ioIndex set to 1 and increment this value by 1 with each call to PBDTGetAPPLAsync until the result code afpItemNotFound is returned in the ioResult field; when called multiple times in this fashion, PBDTGetAPPLAsync returns information about all the application's copies, including the file with the most recent creation date, in arbitrary order.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetAPPLSync

Identifies the application that can open a file with a given creator. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioNamePtr`

On output, a pointer to the application's name.

`ioDTRefNum`

On input, the desktop database reference number of the desktop database containing the specified application.

`ioIndex`

On input, an index into the application list.

`ioTagInfo`

On output, the application's creation date.

`ioFileCreator`

On input, the signature of the application.

`ioAPPLParID`

On output, the application's parent directory.

A single call, with the `ioIndex` field set to 0, finds the application file with the most recent creation date. If you want to retrieve information about all copies of the application with the given signature, start with `ioIndex` set to 1 and increment this value by 1 with each call to `PBDTGetAPPLSync` until the result code `afpItemNotFound` is returned in the `ioResult` field; when called multiple times in this fashion, `PBDTGetAPPLSync` returns information about all the application's copies, including the file with the most recent creation date, in arbitrary order.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetCommentAsync

Retrieves the user comments for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`

On input, a pointer to the name of the file or directory for which you want to retrieve comments.

`ioDTRefNum`

On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDTBuffer`

On input, a pointer to a buffer allocated to hold the comment text. On output, a pointer to the comment text. Allocate a buffer at least 255 bytes in size. The `PBDTGetCommentAsync` function places up to `ioDTReqCount` bytes of the comment into the buffer as a plain text string and places the actual length of the comment in the `ioDTActCount` field.

`ioDTReqCount`

On input, the size of the buffer allocated to hold the comment.

`ioDTActCount`

On output, the comment size.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDirID`
On input, the parent directory of the file or directory.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTGetCommentSync

Retrieves the user comments for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the name of the file or directory for which you want to retrieve comments.

`ioDTRefNum`
On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDTBuffer`
On input, a pointer to a buffer allocated to hold the comment text. On output, a pointer to the comment text. Allocate a buffer at least 255 bytes in size. The `PBDTGetCommentSync` function places up to `ioDTReqCount` bytes of the comment into the buffer as a plain text string and places the actual length of the comment in the `ioDTActCount` field.

`ioDTReqCount`
On input, the size of the buffer allocated to hold the comment.

`ioDTActCount`
On output, the comment size.

`ioDirID`
On input, the parent directory of the file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

FSMegaInfo

Declared In

`Files.h`

PBDTGetIconAsync

Retrieves an icon definition. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBDTGetIconAsync` function returns the bitmap for an icon that represents a file of a given type and creator. For example, to get the icon for a file of file type `'SFWR'` created by the application with a signature of `'WAVE'`, specify these two values in the `ioFileType` and `ioFileCreator` fields.

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`

On input, the desktop database reference number.

`ioTagInfo`

Reserved; on input, this field must be set to 0.

`ioDTBuffer`

On input, a pointer to a buffer to hold the icon's data. On return, a pointer to the bitmap returned in the buffer.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDTReqCount`
: On input, the requested size of the icon's bitmap. Pass the size in bytes of the buffer that you've allocated for the icon's bitmap pointed to by the `ioDTBuffer` field; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. You can use the constants described in "Icon Size Constants" (page 307) to indicate the amount of memory you have provided for the icon's data.

`ioDTActCount`
: On return, the actual size of the icon's bitmap. If this value is larger than the value specified in the `ioDTReqCount` field, only the amount of data allowed by the value in the `ioDTReqCount` field is valid.

`ioIconType`
: On input, the icon type. For a description of the constants which you can use in this field, see "Icon Type Constants" (page 308).

`ioFileCreator`
: On input, the icon's file creator.

`ioFileType`
: On input, the icon's file type.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetIconInfoAsync

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconInfoAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`
: On input, the desktop database reference number.

`ioIndex`
: On input, an index into the icon list.

`ioTagInfo`
: Reserved; on input, this field must be set to 0.

`ioDTActCount`
: On output, the size of the icon's bitmap.

`ioIconType`
: On output, the icon type, including the icon size and color depth. For a description of the values which may be returned in this field, see "Icon Type Constants" (page 308). Ignore any values returned in `ioIconType` that are not listed there; they represent special icons and information used only by the Finder.

`ioFileCreator`
: On input, the icon's file creator.

`ioFileType`
: On output, the icon's file type.

To step through a list of the icon types and file types supported by an application, make repeated calls to `PBDTGetIconInfoAsync`, specifying a creator and an index value in the `ioIndex` field for each call. Set the index to 1 on the first call, and increment it on each subsequent call until the result code `afpItemNotFound` is returned in the `ioResult` field.

To get a list of file types that an application can natively open, you can use the Translation Manager function, `GetFileTypesThatAppCanNativelyOpen`. For a description of this function, see the *Translation Manager Reference* .

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetIconInfoSync

Retrieves an icon type and the associated file type supported by a given creator in the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBDTGetIconInfoSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number.

`ioIndex`
On input, an index into the icon list.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTActCount`
On output, the size of the icon's bitmap.

`ioIconType`
On output, the icon type, including the icon size and color depth. For a description of the values which may be returned in this field, see "Icon Type Constants" (page 308). Ignore any values returned in `ioIconType` that are not listed there; they represent special icons and information used only by the Finder.

`ioFileCreator`
On input, the icon's file creator.

`ioFileType`
On output, the icon's file type.

To step through a list of the icon types and file types supported by an application, make repeated calls to `PBDTGetIconInfoSync`, specifying a creator and an index value in the `ioIndex` field for each call. Set the index to 1 on the first call, and increment it on each subsequent call until the result code `afpItemNotFound` is returned in the `ioResult` field.

To get a list of file types that an application can natively open, you can use the Translation Manager function, `GetFileTypesThatAppCanNativelyOpen`. For a description of this function, see the *Translation Manager Reference* .

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBDTGetIconSync

Retrieves an icon definition. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetIconSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBDTGetIconSync` function returns the bitmap for an icon that represents a file of a given type and creator. For example, to get the icon for a file of file type `'SFWR'` created by the application with a signature of `'WAVE'`, specify these two values in the `ioFileType` and `ioFileCreator` fields.

The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number.

`ioTagInfo`
Reserved; on input, this field must be set to 0.

`ioDTBuffer`
On input, a pointer to a buffer to hold the icon's data. On return, a pointer to the bitmap returned in the buffer.

`ioDTReqCount`
On input, the requested size of the icon's bitmap. Pass the size in bytes of the buffer that you've allocated for the icon's bitmap, pointed to by the `ioDTBuffer` field; this value depends on the icon type. Be sure to allocate enough storage for the icon data; 1024 bytes is the largest amount required for any icon in System 7. You can use the constants described in "Icon Size Constants" (page 307) to indicate the amount of memory you have provided for the icon's data.

`ioDTActCount`
On output, the actual size of the icon's bitmap. If this value is larger than the value specified in the `ioDTReqCount` field, only the amount of data allowed by `ioDTReqCount` is valid.

`ioIconType`
On input, the icon type. For a description of the constants which you can use in this field, see "Icon Type Constants" (page 308).

`ioFileCreator`
On input, the icon's file creator.

`ioFileType`
On input, the icon's file type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetInfoAsync

Determines information about the location and size of the desktop database on a particular volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetInfoAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion functions, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioVRefNum`

On output, the volume reference number of the volume where the database files are stored.

`ioDTRefNum`

On input, the desktop database reference number of the database which you wish to obtain information about.

`ioIndex`

On output, the number of files comprising the desktop database on the volume.

`ioDirID`

On output, the parent directory ID of the desktop database.

`ioDTLgLen`

On output, the logical length of the database files (the sum of the logical lengths of the files that constitute the desktop database for a given volume).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDTPyLen`
: On output, the physical length of the database files (the sum of the physical lengths of the files that constitute the desktop database for a given volume).

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetInfoSync

Determines information about the location and size of the desktop database on a particular volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetInfoSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioVRefNum`
: On output, the volume reference number of the volume where the database files are stored.

`ioDTRefNum`
: On input, the desktop database reference number of the database which you wish to obtain information about.

`ioIndex`
: On output, the number of files comprising the desktop database on the volume.

`ioDirID`
: On output, the parent directory ID of the desktop database.

`ioDTLgLen`
: On output, the logical length of the database files (the sum of the logical lengths of the files that constitute the desktop database for a given volume).

`ioDTPyLen`
: On output, the physical length of the database files (the sum of the physical lengths of the files that constitute the desktop database for a given volume).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTGetPath

Gets the reference number of the specified desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTGetPath (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioNamePtr`

On input, a pointer to the name of the volume associated with the desktop database or the full pathname of the desktop database.

`ioVRefNum`

On input, the volume reference number of the volume associated with the desktop database.

`ioDTRefNum`

On output, the desktop database reference number, which represents the access path to the database. You cannot use the desktop reference number as a file reference number in any File Manager functions other than the desktop database functions. If `PBDTGetPath` fails, it sets this field to 0.

If the desktop database is not already open, `PBDTGetPath` opens it and then returns the reference number. If the desktop database doesn't exist, `PBDTGetPath` creates it .

Special Considerations

`PBDTGetPath` allocates memory in the system heap; do not call it at interrupt time.

This function executes synchronously only.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Not available to 64-bit applications.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

PBDTOpenInform

Gets the reference number of the specified desktop database, reporting whether the desktop database was empty when it was opened. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTOpenInform (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the name of the volume associated with the desktop database or the full pathname of the desktop database.

`ioVRefNum`
On input, the volume reference number of the volume associated with the desktop database.

`ioDTRefNum`
On output, the desktop database reference number, which represents the access path to the database. You cannot use the desktop reference number as a file reference number in any File Manager functions other than the desktop database functions. If `PBDTOpenInform` fails, it sets this field to 0.

`ioTagInfo`
On output, the return flag (in the low bit of this field). If the desktop database was just created in response to `PBDTOpenInform` (and is therefore empty), `PBDTOpenInform` sets the low bit in this field to 0. If the desktop database had been created before you called `PBDTOpenInform`, `PBDTOpenInform` sets the low bit in this field to 1.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

This function executes synchronously only.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBDTRemoveAPPLAsync

Removes an application from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveAPPLAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a desktop database parameter block. See DTPBRec (page 199) for a description of the DTPBRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). When called on an HFS CD volume, PBDTRemoveAPPL returns an afpItemNotFound error, instead of the expected volume locked error (wPrErr).

Discussion
The PBDTRemoveAPPLAsync function removes the mapping information for an application from the database specified in the ioDTRefNum field. You can call PBDTRemoveAPPLAsync even if the application is not present on the volume.

The relevant fields of the parameter block for this function are:

ioCompletion
On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult
On output, the result code of the function. See "File Manager Result Codes".

ioNamePtr
On input, a pointer to the application's name.

ioDTRefNum
On input, the desktop database reference number of the desktop database containing the application.

ioDirID
On input, the application's parent directory.

ioFileCreator
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBDTRemoveAPPLSync

Removes an application from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTRemoveAPPLSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBDTRemoveAPPLSync` function removes the mapping information for an application from the database specified in the `ioDTRefNum` field. You can call `PBDTRemoveAPPLSync` even if the application is not present on the volume.

The relevant fields of the parameter block for this function are:

`ioNamePtr`
On input, a pointer to the application's name.

`ioDTRefNum`
On input, the desktop database reference number of the desktop database containing the application.

`ioDirID`
On input, the application's parent directory.

`ioFileCreator`
On input, the application's signature.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTRemoveCommentAsync

Removes a user comment associated with a file or directory from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBDTRemoveCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`

On input, a pointer to the filename or directory name.

`ioDTRefNum`

On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDirID`

On input, the parent directory ID of the file or directory.

You cannot remove a comment if the file or directory it is associated with is not present on the volume. If no comment was stored for the file, `PBDTRemoveCommentAsync` returns an error.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTRemoveCommentSync

Removes a user comment associated with a file or directory from the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBDTRemoveCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block for this function are:

`ioNamePtr`

On input, a pointer to the filename or directory name.

`ioDTRefNum`

On input, the desktop database reference number of the database in which the specified file or directory is found.

`ioDirID`

On input, the parent directory ID of the file or directory.

You cannot remove a comment if the file or directory it is associated with is not present on the volume. If no comment was stored for the file, `PBDTRemoveCommentSync` returns an error.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDTResetAsync

Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTResetAsync (
   DTPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBDTResetAsync` function removes all icons, application mappings, and comments from the desktop database specified in the `ioDTRefNum` field. You can call `PBDTResetAsync` only when the database is open. It remains open after the data is cleared. Your application should not call `PBDTResetAsync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to clear.

`ioIndex`
Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTResetSync

Removes information from the desktop database. Unless you are manipulating the desktop database in the absence of the Finder, you should never use this function. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTResetSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The `PBDTResetSync` function removes all icons, application mappings, and comments from the desktop database specified in the `ioDTRefNum` field. You can call `PBDTResetSync` only when the database is open. It remains open after the data is cleared. Your application should not call `PBDTResetSync` unless absolutely necessary.

The relevant fields of the parameter block for this function are:

`ioDTRefNum`
On input, the desktop database reference number of the desktop database to clear.

`ioIndex`
Reserved; on input, this field must be set to 0.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTSetCommentAsync

Adds a user comment for a file or a directory to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTSetCommentAsync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. See "File Manager Result Codes".

`ioNamePtr`
On input, a pointer to the name of the file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDTRefNum`
: On input, the desktop database reference number for the desktop database to which to add the user comment.

`ioDTBuffer`
: On input, a pointer to the buffer containing the comment text. Put the comment in the buffer as a plain text string.

`ioDTReqCount`
: On input, the length of the buffer (in bytes) containing the comment text. The maximum length of a comment is 200 bytes; longer comments are truncated. Since the comment is a plain text string and not a Pascal string, the File Manager relies on the value in the `ioDTReqCount` field for determining the length of the buffer.

`ioDirID`
: On input, the parent directory ID of the file or directory.

If the specified object already has a comment in the database, the new comment replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDTSetCommentSync

Adds a user comment for a file or a directory to the desktop database. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBDTSetCommentSync (
   DTPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a desktop database parameter block. See `DTPBRec` (page 199) for a description of the `DTPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block for this function are:

`ioNamePtr`
: On input, a pointer to the name of the file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDTRefNum`
: On input, the desktop database reference number for the desktop database to which to add the user comment.

`ioDTBuffer`
: On input, a pointer to the buffer containing the comment text. Put the comment in the buffer as a plain text string.

`ioDTReqCount`
: On input, the length of the buffer containing the comment text, in bytes. The maximum length of a comment is 200 bytes; longer comments are truncated. Since the comment is a plain text string and not a Pascal string, the File Manager relies on the value in the `ioDTReqCount` field for determining the length of the buffer.

`ioDirID`
: On input, the parent directory ID of the file or directory.

If the specified object already has a comment in the database, the new comment replaces the old.

Special Considerations

All of the desktop database functions may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBExchangeFilesAsync

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `PBExchangeObjectsAsync` (page 131) instead.)

```
OSErr PBExchangeFilesAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `FIDParam` (page 204) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the name of the first file to swap.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDestNamePtr`
On input, a pointer to the name of the second file to swap.

`ioDestDirID`
On input, the second file's parent directory ID.

`ioSrcDirID`
On input, the first file's parent directory ID.

Typically, you use `PBExchangeFilesAsync` after creating a new file during a safe save. The `PBExchangeFilesAsync` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

The `PBExchangeFilesAsync` function works on either open or closed files. `PBExchangeFilesAsync` swaps the data in two files by changing some of the information in the volume catalog. If either file is open, `PBExchangeFilesAsync` updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access privileges as opening both files for writing.

The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `PBExchangeFilesAsync` to preserve the file ID when updating an existing file, in case the file is being tracked through its file ID. The `PBExchangeFilesAsync` function does not require that file ID references exist for the files being exchanged.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the `FSExchangeObjects` (page 59) , `PBExchangeObjectsSync` (page 132) , or `PBExchangeObjectsAsync` (page 131) function.

Special Considerations

Your application will have to swap any open reference numbers to the two files because the file's name and parent directory ID are exchanged in the file control blocks.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (`fsRdWrPerm`) to both files before calling `PBExchangeFilesAsync`. Exclusive read/write access to both files will ensure that `PBExchangeFilesAsync` doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

`PBExchangeFilesAsync` does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling `PBExchangeFilesAsync` ensures that the files are unlocked because locked files cannot be opened with write access.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBExchangeFilesSync

Exchanges the data stored in two files on the same volume. (Deprecated in Mac OS X v10.4. Use `PBExchangeObjectsSync` (page 132) instead.)

```
OSErr PBExchangeFilesSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `FIDParam` (page 204) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the first file to swap.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDestNamePtr`
On input, a pointer to the name of the second file to swap.

`ioDestDirID`
On input, the second file's parent directory ID.

`ioSrcDirID`
On input, the first file's parent directory ID.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Typically, you use `PBExchangeFilesSync` after creating a new file during a safe save. The `PBExchangeFilesSync` function changes the fields in the catalog entries that record the location of the data and the modification dates. It swaps both the data forks and the resource forks.

The `PBExchangeFilesSync` function works on either open or closed files. `PBExchangeFilesSync` swaps the data in two files by changing some of the information in the volume catalog. If either file is open, `PBExchangeFilesSync` updates any file control blocks associated with the file. Exchanging the contents of two files requires essentially the same access privileges as opening both files for writing.

The following fields in the catalog entries for the files are exchanged:

- `ioFlStBlk`
- `ioFlLgLen`
- `ioFlPyLen`
- `ioFlRStBlk`
- `ioFlRLgLen`
- `ioFlRPyLen`
- `ioFlMdDat`

In the file control blocks, the `fcbFlNum`, `fcbDirID`, and `fcbCName` fields are exchanged.

You should use `PBExchangeFilesSync` to preserve the file ID when updating an existing file, in case the file is being tracked through its file ID. The `PBExchangeFilesSync` function does not require that file ID references exist for the files being exchanged.

To exchange the contents of files with named forks other than the data and resource forks, or of files larger than 2 GB, use the `FSExchangeObjects` (page 59) , `PBExchangeObjectsSync` (page 132) , or `PBExchangeObjectsAsync` (page 131) function.

Special Considerations

Your application will have to swap any open reference numbers to the two files because the file's name and parent directory ID are exchanged in the file control blocks.

Because other programs may have access paths open to one or both of the files exchanged, your application should have exclusive read/write access permission (`fsRdWrPerm`) to both files before calling `PBExchangeFilesSync`. Exclusive read/write access to both files will ensure that `PBExchangeFilesSync` doesn't affect another application because it prevents other applications from obtaining write access to one or both of the files exchanged.

`PBExchangeFilesSync` does not respect the file-locked attribute; it will perform the exchange even if one or both of the files are locked. Obtaining exclusive read/write access to both files before calling `PBExchangeFilesSync` ensures that the files are unlocked because locked files cannot be opened with write access.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBFlushFileAsync

Writes the contents of a file's access path buffer to the disk. (Deprecated in Mac OS X v10.4. Use `PBFlushForkAsync` (page 133) instead.)

```
OSErr PBFlushFileAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioRefNum`

On input, a file reference number for the file to flush.

After writing the contents of the file to the volume, the `PBFlushFileAsync` function updates the file's entry in the volume catalog.

In the event of a system crash, all cached data not yet written to disk is lost. If you have made changes to space that already exists within a file (you are overwriting existing data before the file's end-of-file), you must use `PBFlushFileAsync` to ensure that everything written to the file will be written to disk. If you flush the fork's cached blocks using `PBFlushFileAsync`, the only possible data loss in a system crash will be the file's modification date.

You do not, however, need to use `PBFlushFileAsync` to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

`PBFlushFileSync` flushes an open fork's dirty cached blocks, but may not flush catalog information associated with the file. To flush catalog information, call `FlushVol` (page 502) , or one of the related parameter block calls, `PBFlushVolSync` (page 510) and `PBFlushVolAsync` (page 509).

To update a file larger than 2GB, or a named fork other than the data and resource forks, you must use the `FSFlushFork` (page 64) function, or one of the corresponding parameter block calls, `PBFlushForkSync` (page 133) and `PBFlushForkAsync` (page 133).

Special Considerations

Some information stored on the volume won't be correct until `PBFlushVolAsync` is called.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBFlushFileSync

Writes the contents of a file's access path buffer to the disk. (Deprecated in Mac OS X v10.4. Use PBFlushForkSync (page 133) instead.)

```
OSErr PBFlushFileSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 248) variant of the basic File Manager parameter block. See ParamBlockRec (page 252) for a description of the ParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant field of the parameter block is:

ioRefNum
On input, a file reference number for the file to flush.

After writing the contents of the file to the volume, the PBFlushFileSync function updates the file's entry in the volume catalog.

In the event of a system crash, all cached data not yet written to disk is lost. If you have made changes to space that already exists within a file (you are overwriting existing data before the file's end-of-file), you must use PBFlushFileSync to ensure that everything written to the file will be written to disk. If you flush the fork's cached blocks using PBFlushFileSync, the only possible data loss in a system crash will be the file's modification date.

You do not, however, need to use PBFlushFileSync to flush a file fork before it is closed; the file is automatically flushed when it is closed and all cache blocks associated with it are removed from the cache.

PBFlushFileSync flushes an open fork's dirty cached blocks, but may not flush catalog information associated with the file. To flush catalog information, call FlushVol (page 502) , or one of the related parameter block calls, PBFlushVolSync (page 510) and PBFlushVolAsync (page 509).

To update a file larger than 2GB, or a named fork other than the data and resource forks, you must use the FSFlushFork (page 64) function, or one of the corresponding parameter block calls, PBFlushForkSync (page 133) and PBFlushForkAsync (page 133).

Special Considerations

Some information stored on the volume won't be correct until PBFlushVolSync is called.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBGetCatInfoAsync

Returns catalog information about a file or directory. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoAsync` (page 135) instead.)

```
OSErr PBGetCatInfoAsync (
   CInfoPBPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 187) for a description of the `CInfoPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBGetCatInfoAsync` function returns information about a file or directory, depending on the values you specify in the `ioFDirIndex`, `ioNamePtr`, `ioVRefNum`, and `ioDirID` or `ioDrDirID` fields. If you need to determine whether the information returned is for a file or a directory, you can test bit 4 of the `ioFlAttrib` field; if that bit is set, the information returned describes a directory.

The `PBGetCatInfoAsync` function selects a file or directory according to these rules:

- If the value of `ioFDirIndex` is positive, `ioNamePtr` is not used as an input parameter and `PBGetCatInfoAsync` returns information about the file or directory whose directory index is `ioFDirIndex` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0). If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the file or directory name will be returned.
- If the value of `ioFDirIndex` is 0, `PBGetCatInfoAsync` returns information about the file or directory specified by `ioNamePtr` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (again, this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0).
- If the value of `ioFDirIndex` is negative, `PBGetCatInfoAsync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDrDirID` field. If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the directory name will be returned.

With files, `PBGetCatInfoAsync` is similar to `PBHGetFInfoAsync` (page 442) but returns some additional information. If the object is a file, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
: On input, a pointer to a pathname. On output, the name of the file is returned in this field, if the file is open. If you do not want the name of the file returned, pass `NULL` in this field.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioFRefNum`
: On output, a file reference number. If the file is open, the reference number of the first access path found is returned here .

`ioFDirIndex`
: On input, a directory index.

`ioFlAttrib`
: On output, the file attributes. See "File Attribute Constants" (page 300) for the meaning of the file attributes.

`ioFlFndrInfo`
: On output, information used by the Finder.

`ioDirID`
: On input, a directory ID. On output, the file ID. You might need to save the value of `ioDirID` before calling `PBGetCatInfoAsync` if you make subsequent calls with the same parameter block.

`ioFlStBlk`
: On output, the first allocation block of the data fork.

`ioFlLgLen`
: On output, the logical size (the logical end-of-file) of the data fork, in bytes.

`ioFlPyLen`
: On output, the physical size (the physical end-of-file) of the data fork, in bytes.

`ioFlRStBlk`
: On output, the first allocation block of the resource fork.

`ioFlRLgLen`
: On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
: On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
: On output, the date and time of the file's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioFlCrDat` field to 0.

`ioFlMdDat`
: On output, the date and time of the file's last modification.

`ioFlBkDat`
: On output, the date and time of the file's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioFlBkDat` field to 0.

`ioFlXFndrInfo`
: On output, additional information used by the Finder.

`ioFlParID`
: On output, the directory ID of the file's parent directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioFlClpSiz`
On output, the file's clump size.

You can also use `PBGetCatInfoAsync` to determine whether a file has a file ID reference. The value of the file ID is returned in the `ioDirID` field. Because that parameter could also represent a directory ID, call `PBResolveFileIDRefAsync` (page 534) to see if the value is a real file ID. If you want to determine whether a file ID reference exists for a file and create one if it doesn't, use `PBCreateFileIDRefAsync` (page 505) , which will either create a file ID or return `fidExists`.

If the object is a directory, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname. On output, a pointer to the directory name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the directory attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field. The bits in this field for directories are read-only. You cannot alter directory attributes by setting these bits using the functions `PBSetCatInfoSync` (page 481) or `PBSetCatInfoAsync` (page 480). Instead, you can call the `PBHSetFLockSync` (page 471) and `PBHRstFLockSync` (page 468) functions to lock and unlock a directory, and the `PBShareSync` (page 490) and `PBUnshareSync` (page 494) functions to enable and disable file sharing on local directories.

`ioACUser`
On output, the directory access rights. The `PBGetCatInfoAsync` function returns the information in this field only for shared volumes. As a result, you should set this field to 0 before calling `PBGetCatInfoAsync`.`PBGetCatInfoAsync` does not return the blank access privileges bit in this field; to determine whether a directory has blank access privileges, use the `PBHGetDirAccessAsync` (page 515) function. See "User Privileges Constants" (page 316) for a description of the constants that may be returned in this field.

`ioDrUsrWds`
On output, information used by the Finder.

`ioDrDirID`
On input, if you wish to obtain information about a specific directory, that directory's ID. Otherwise, if the object returned is a directory, this field contains the directory ID on output.

`ioDrNmFls`
On output, the number of files in the directory.

`ioDrCrDat`
On output, the date and time of the directory's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioDrCrDat` field to 0.

`ioDrMdDat`
On output, the date and time of the directory's last modification.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDrBkDat`
: On output, the date and time of the directory's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioDrBkDat` field to 0.

`ioDrFndrInfo`
: On output, additional information used by the Finder.

`ioDrParID`
: On output, the directory ID of the directory's parent directory.

To get information on a file or directory with named forks, or on a file larger than 2GB, use one of the `FSGetCatalogInfo` (page 67) , `PBGetCatalogInfoSync` (page 139) , or `PBGetCatalogInfoAsync` (page 135) functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetCatInfoSync

Returns catalog information about a file or directory. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoSync` (page 139) instead.)

```
OSErr PBGetCatInfoSync (
   CInfoPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 187) for a description of the `CInfoPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBGetCatInfoSync` function returns information about a file or directory, depending on the values you specify in the `ioFDirIndex`, `ioNamePtr`, `ioVRefNum`, and `ioDirID` or `ioDrDirID` fields. If you need to determine whether the information returned is for a file or a directory, you can test bit 4 of the `ioFlAttrib` field; if that bit is set, the information returned describes a directory.

The `PBGetCatInfoSync` function selects a file or directory according to these rules:

- If the value of `ioFDirIndex` is positive, `ioNamePtr` is not used as an input parameter and `PBGetCatInfoSync` returns information about the file or directory whose directory index is `ioFDirIndex` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and `ioDirID` is 0). If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the file or directory name will be returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

- If the value of `ioFDirIndex` is 0, `PBGetCatInfoSync` returns information about the file or directory specified by `ioNamePtr` in the directory specified by `ioDirID` (or `ioDrDirID`) on the volume specified by `ioVRefNum` (again, this will be the root directory if `ioVRefNum` is a volume reference number or a drive number and ioDirID is 0).
- If the value of `ioFDirIndex` is negative, `PBGetCatInfoSync` ignores the `ioNamePtr` field and returns information about the directory specified in the `ioDrDirID` field. If `ioNamePtr` is not `NULL`, then it must point to a `Str31` buffer where the directory name will be returned.

With files, `PBGetCatInfoSync` is similar to `PBHGetFInfoSync` (page 444) but returns some additional information. If the object is a file, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname. On output, the name of the file is returned in this field, if the file is open. If you do not want the name of the file returned, pass `NULL` in this field.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFRefNum`
On output, a file reference number. If the file is open, the reference number of the first access path found is returned here.

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the file attributes. See "File Attribute Constants" (page 300) for the meaning of the file attributes.

`ioFlFndrInfo`
On output, information used by the Finder.

`ioDirID`
On input, a directory ID. On output, the file ID. You might need to save the value of `ioDirID` before calling `PBGetCatInfoSync` if you make subsequent calls with the same parameter block.

`ioFlStBlk`
On output, the first allocation block of the data fork.

`ioFlLgLen`
On output, the logical size (the logical end-of-file) of the data fork, in bytes.

`ioFlPyLen`
On output, the physical size (the physical end-of-file) of the data fork, in bytes.

`ioFlRStBlk`
On output, the first allocation block of the resource fork.

`ioFlRLgLen`
On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
On output, the date and time of the file's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioFlCrDat` field to 0.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioFlMdDat`
On output, the date and time of the file's last modification.

`ioFlBkDat`
On output, the date and time of the file's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioFlBkDat` field to 0.

`ioFlXFndrInfo`
On output, additional information used by the Finder.

`ioFlParID`
On output, the directory ID of the file's parent directory.

`ioFlClpSiz`
On output, the file's clump size.

You can also use `PBGetCatInfoSync` to determine whether a file has a file ID reference. The value of the file ID is returned in the `ioDirID` field. Because that parameter could also represent a directory ID, call `PBResolveFileIDRefSync` (page 535) to see if the value is a real file ID. If you want to determine whether a file ID reference exists for a file and create one if it doesn't, use `PBCreateFileIDRefSync` (page 506) , which will either create a file ID or return `fidExists`.

If the object is a directory, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname. On output, a pointer to the directory's name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFDirIndex`
On input, a directory index.

`ioFlAttrib`
On output, the directory attributes. See "File Attribute Constants" (page 300) for the meaning of the bits in this field. The bits in this field for directories are read-only. You cannot alter directory attributes by setting these bits using the functions `PBSetCatInfoSync` (page 481) or `PBSetCatInfoAsync` (page 480). Instead, you can call the `PBHSetFLockSync` (page 471) and `PBHRstFLockSync` (page 468) functions to lock and unlock a directory, and the `PBShareSync` (page 490) and `PBUnshareSync` (page 494) functions to enable and disable file sharing on local directories.

`ioACUser`
On output, the directory access rights. The `PBGetCatInfoSync` function returns the information in this field only for shared volumes. As a result, you should set this field to 0 before calling `PBGetCatInfoSync`. `PBGetCatInfoSync` does not return the blank access privileges bit in this field; to determine whether a directory has blank access privileges, use the `PBHGetDirAccessSync` (page 516) function. See "User Privileges Constants" (page 316) for a description of the constants that may be returned here.

`ioDrUsrWds`
On output, information used by the Finder.

`ioDrDirID`
On input, if you wish to obtain information about a specific directory, that directory's ID. Otherwise, if the object returned is a directory, this field contains the directory ID on output.

`ioDrNmFls`
On output, the number of files in the directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDrCrDat`
: On output, the date and time of the directory's creation. Note that file systems other than AFP, HFS and HFS Plus do not generally support creation dates. For file systems which do not support creation dates, the File Manager sets the `ioDrCrDat` field to 0.

`ioDrMdDat`
: On output, the date and time of the directory's last modification.

`ioDrBkDat`
: On output, the date and time of the directory's last backup. Note that file systems other than AFP, HFS and HFS Plus do not generally support backup dates. For file systems which do not support backup dates, the File Manager sets the `ioDrBkDat` field to 0.

`ioDrFndrInfo`
: On output, additional information used by the Finder.

`ioDrParID`
: On output, the directory ID of the directory's parent directory.

To get information on a file or directory with named forks, or on a file larger than 2GB, use one of the `FSGetCatalogInfo` (page 67) , `PBGetCatalogInfoSync` (page 139) , or `PBGetCatalogInfoAsync` (page 135) functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
MFSLives
ThreadsExporter
ThreadsImporter
ThreadsImportMovie

Declared In
`Files.h`

PBGetEOFAsync

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBGetForkSizeAsync` (page 145) instead.)

```
OSErr PBGetEOFAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioRefNum`
: On input, a file reference number for the open file.

`ioMisc`
: On output, the logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you'll need to coerce the value to a long integer to interpret the value correctly.

To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the `FSGetForkSize` (page 73) function, or one of the corresponding parameter block functions, `PBGetForkSizeSync` (page 145) and `PBGetForkSizeAsync` (page 145).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetEOFSync

Determines the current logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBGetForkSizeSync` (page 145) instead.)

```
OSErr PBGetEOFSync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
: On input, a file reference number for the open file.

`ioMisc`
: On output, a pointer to the logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you'll need to coerce the value to a long integer to interpret the value correctly.

To determine the size of a named fork other than the data or resource forks, or of a fork larger than 2 GB, use the `FSGetForkSize` (page 73) function, or one of the corresponding parameter block functions, `PBGetForkSizeSync` (page 145) and `PBGetForkSizeAsync` (page 145).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetFCBInfoAsync

Gets information about an open file from the file control block. (Deprecated in Mac OS X v10.4. Use `PBGetForkCBInfoAsync` (page 140) instead.)

```
OSErr PBGetFCBInfoAsync (
   FCBPBPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a file control block parameter block. See `FCBPBRec` (page 202) for a description of the `FCBPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname. You should pass a pointer to a `Str31` value if you want the name of the file returned. If you pass `NULL`, no filename is returned. On output, if `PBGetFCBInfoAsync` executes successfully, a pointer to the name of the specified open file.

`ioVRefNum`
On input, a volume specification. If you specify a valid index number in the `ioFCBIndx` field, the File Manager returns information on the file having that index in the FCB buffer on the volume specified in this field. This field may contain a drive number or volume reference number. If the value of `ioVRefNum` is 0, all open files are indexed; otherwise, only open files on the specified volume are indexed.

`ioRefNum`
On input, if the `ioFCBIndx` field is 0, the file reference number of the file to get information about. If the value of `ioFCBIndx` is positive, the `ioRefNum` field is ignored on input and contains the file reference number on output.

`ioFCBIndx`
On input, an index. If the value of `ioFCBIndx` is positive, the File Manager returns information about the file whose index in the FCB buffer is `ioFCBIndx` and that is located on the volume specified in

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

the `ioVRefNum` field. If the value of `ioFCBIndx` is 0, the File Manager returns information about the file whose file reference number is specified by the `ioRefNum` field.

`ioFCBFlNm`
On output, the file ID.

`ioFCBFlags`
On output, file status flags. See "FCB Flags" (page 293) for a description of the bits in this field.

`ioFCBStBlk`
On output, the first allocation block of the file.

`ioFCBEOF`
On output, the logical size (the logical end-of-file) of the file.

`ioFCBPLen`
On output, the physical size (the physical end-of-file) of the file.

`ioFCBCrPs`
On output, the position of the file mark.

`ioFCBVRefNum`
On output, the volume reference number.

`ioFCBClpSiz`
On output, the file clump size.

`ioFCBParID`
On output, the directory ID of the file's parent directory.

To get information about a fork control block, use one of the functions, `FSGetForkCBInfo` (page 71) , `PBGetForkCBInfoSync` (page 142) , or `PBGetForkCBInfoAsync` (page 140).

Special Considerations

On OS X, the value returned by `PBGetFCBInfoAsync` in the `ioFCBPLen` field may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `PBGetFCBInfoAsync` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetFCBInfoSync

Gets information about an open file from the file control block. (Deprecated in Mac OS X v10.4. Use `PBGetForkCBInfoSync` (page 142) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBGetFCBInfoSync (
   FCBPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a file control block parameter block. See `FCBPBRec` (page 202) for a description of the `FCBPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname. You should pass a pointer to a `Str31` value if you want the name of the file returned. If you pass `NULL`, no filename is returned. On output, if `PBGetFCBInfoSync` executes successfully, a pointer to the name of the specified open file.

`ioVRefNum`

On input, a volume specification. If you specify a valid index number in the `ioFCBIndx` field, the File Manager returns information on the file having that index in the FCB buffer on the volume specified in this field. This field may contain a drive number or volume reference number. If the value of `ioVRefNum` is 0, all open files are indexed; otherwise, only open files on the specified volume are indexed.

`ioRefNum`

On input, if the ioFCBIndx field is 0, the file reference number of the file to get information about. If the value of `ioFCBIndx` is positive, the `ioRefNum` field is ignored on input and contains the file reference number on output.

`ioFCBIndx`

On input, an index. If the value of `ioFCBIndx` is positive, the File Manager returns information about the file whose index in the FCB buffer is `ioFCBIndx` and that is located on the volume specified in the `ioVRefNum` field. If the value of `ioFCBIndx` is 0, the File Manager returns information about the file whose file reference number is specified by the `ioRefNum` field.

`ioFCBFlNm`

On output, the file ID.

`ioFCBFlags`

On output, file status flags. See "FCB Flags" (page 293) for a description of the bits in this field.

`ioFCBStBlk`

On output, the first allocation block of the file.

`ioFCBEOF`

On output, the logical size (the logical end-of-file) of the file.

`ioFCBPLen`

On output, the physical size (the physical end-of-file) of the file.

`ioFCBCrPs`

On output, the current position of the file mark.

`ioFCBVRefNum`

On output, the volume reference number.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioFCBClpSiz`
On output, the file clump size.

`ioFCBParID`
On output, the directory ID of the file's parent directory.

To get information about a fork control block, use one of the functions, `FSGetForkCBInfo` (page 71) , `PBGetForkCBInfoSync` (page 142) , or `PBGetForkCBInfoAsync` (page 140).

Special Considerations

On OS X, the value returned by `PBGetFCBInfoSync` in the `ioFCBPLen` field may differ from the physical file length reported by `FSGetCatalogInfo`, `PBGetCatInfo`, and related functions. When a write causes a file to grow in size, the physical length reported by `FSGetCatalogInfo` and similar calls increases by the clump size, which is a multiple of the allocation block size. However, the physical length returned by `PBGetFCBInfoSync` changes according to the allocation block size and the file lengths returned by the respective functions get out of sync.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
MoreOSL
qtinfo
qtinfo.win

Declared In
`Files.h`

PBGetForeignPrivsAsync

Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetForeignPrivsAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBGetForeignPrivsSync

Determines the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetForeignPrivsSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBGetFPosAsync

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use `PBGetForkPositionAsync` (page 143) instead.)

```
OSErr PBGetFPosAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information about completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioRefNum`

On input, the file reference number of an open file.

`ioPosOffset`

On output, the current position of the mark. The value returned in `ioPosOffset` is zero-based. Thus, a call to `PBGetFPosAsync` returns 0 if you call it when the file mark is positioned at the beginning of the file. The `ioReqCount`, `ioActCount`, and `ioPosMode` fields of the parameter block are all set to 0 on output. To determine the current position of a named fork, or of a fork larger than 2GB, use

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

the FSGetForkPosition (page 72) function, or one of the corresponding parameter block calls, PBGetForkPositionSync (page 144) and PBGetForkPositionAsync (page 143).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBGetFPosSync

Returns the current position of the file mark. (Deprecated in Mac OS X v10.4. Use PBGetForkPositionSync (page 144) instead.)

```
OSErr PBGetFPosSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the IOParam (page 248) variant of the basic File Manager parameter block. See ParamBlockRec (page 252) for a description of the ParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ioRefNum
On input, the file reference number of an open file.

ioPosOffset
On output, the current position of the mark. The value returned in ioPosOffset is zero-based. Thus, a call to PBGetFPosSync returns 0 if you call it when the file mark is positioned at the beginning of the file.

The ioReqCount, ioActCount, and ioPosMode fields of the parameter block are all set to 0 on output.

To determine the current position of a named fork, or of a fork larger than 2GB, use the FSGetForkPosition (page 72) function, or one of the corresponding parameter block calls, PBGetForkPositionSync (page 144) and PBGetForkPositionAsync (page 143).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBGetUGEntryAsync

Gets a user or group entry from the list of User and Group names and IDs on the local file server. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetUGEntryAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBGetUGEntrySync

Gets a user or group entry from the list of User and Group names and IDs on a local file server. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetUGEntrySync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBGetXCatInfoAsync

Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetXCatInfoAsync (
   XCInfoPBPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetXCatInfoSync

Returns the short name (MS-DOS format name) and the ProDOS information for a file or directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBGetXCatInfoSync (
   XCInfoPBPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHCreateAsync

Creates a new file. (Deprecated in Mac OS X v10.4. Use `PBCreateFileUnicodeAsync` (page 124) instead.)

```
OSErr PBHCreateAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion functions, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
On input, a pointer to the name for the new file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the directory ID of the parent directory of the new file.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

The `PBHCreateAsync` function creates both forks of the file the new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time. If the file created isn't temporary (that is, if it will exist after the user quits the application), the application should call `PBHSetFInfoAsync` (page 469) , after the call to `PBHCreateAsync`, to fill in the information needed by the Finder.

Files created using `PBHCreateAsync` are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, `FSpOpenDF` (page 354) , `HOpenDF` (page 368) , `PBHOpenDFSync` (page 460) or `PBHOpenDFAsync` (page 458).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager procedures `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 54) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 125) and `PBCreateFileUnicodeAsync` (page 124).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHCreateSync

Creates a new file. (Deprecated in Mac OS X v10.4. Use `PBCreateFileUnicodeSync` (page 125) instead.)

```
OSErr PBHCreateSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name for the new file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the directory ID of the parent directory of the new file.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

The `PBHCreateSync` function creates both the data and resource fork of the file the new file is unlocked and empty. The date and time of its creation and last modification are set to the current date and time. If the file created isn't temporary (that is, if it will exist after the user quits the application), the application should call `PBHSetFInfoSync` (page 470) after the call to `PBHCreateSync` to fill in the information needed by the Finder.

Files created using `PBHCreateSync` are not automatically opened. If you want to write data to the new file, you must first open the file using one of the file access functions, `FSpOpenDF` (page 354) , `HOpenDF` (page 368) , `PBHOpenDFSync` (page 460) or `PBHOpenDFAsync` (page 458).

The resource fork of the new file exists but is empty. You'll need to call one of the Resource Manager procedures `HCreateResFile` or `FSpCreateResFile` to create a resource map in the file before you can open it (by calling one of the Resource Manager functions `HOpenResFile` or `FSpOpenResFile`).

To create a file with a Unicode filename, use the function `FSCreateFileUnicode` (page 54) , or one of the corresponding parameter block calls, `PBCreateFileUnicodeSync` (page 125) and `PBCreateFileUnicodeAsync` (page 124).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHDeleteAsync

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `PBDeleteObjectAsync` (page 130) instead.)

```
OSErr PBHDeleteAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. If you attempt to delete an open file or a non-empty directory, `PBHDeleteAsync` returns the result code `fBsyErr`. `PBHDeleteAsync` also returns `fBsyErr` if you attempt to delete a directory that has an open working directory associated with it.

`ioNamePtr`
: On input, a pointer to the name of the file or directory to delete.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
: On input, the directory ID of the parent directory of the file or directory to delete.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that file ID reference is also removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHDeleteSync

Deletes a file or directory. (Deprecated in Mac OS X v10.4. Use `PBDeleteObjectSync` (page 130) instead.)

```
OSErr PBHDeleteSync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
: A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). If you attempt to delete an open file or a non-empty directory, `PBHDeleteSync` returns the result code `fBsyErr`. `PBHDeleteSync` also returns `fBsyErr` if you attempt to delete a directory that has an open working directory associated with it.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the file or directory to delete.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the directory ID of the parent directory of the file or directory to delete.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the specified target is a file, both the data and the resource fork of the file are deleted. In addition, if a file ID reference for the specified file exists, that file ID reference is also removed. A file must be closed before you can delete it. Similarly, you cannot delete a directory unless it's empty.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetFInfoAsync

Obtains information about a file. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoAsync` (page 135) instead.)

```
OSErr PBHGetFInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
: On input, a pointer to a pathname. If the value of the `ioFDirIndex` field is negative or 0, `PBHGetFInfoAsync` returns information about the file in the volume specified by the reference number in the `ioVRefNum` field and having the name given here. On output, a pointer to the name of the file, if the file is open. If you do not wish the name returned, pass `NULL` here.

`ioVRefNum`
: On input, a volume reference number or drive number for the volume containing the file, or 0 for the default volume.

`ioFRefNum`
: On output, the reference number of the first access path found, if the file is open and if the `ioFDirIndex` field is negative or 0; if the `ioFDirIndex` field is positive...

`ioFDirIndex`
: On input, a directory index. If this value is positive, the function returns information about the file having the directory index specified here, on the volume specified in the `ioVRefNum` field and in the directory specified in the `ioDirID` field. If this value is negative or 0, the function returns information about the file on the specified volume, having the name pointed to in the `ioNamePtr` field.

`ioFlAttrib`
: On output, the file attributes. See "File Attribute Constants" (page 300) for a description of the file attributes.

`ioFlFndrInfo`
: On output, Finder information about the file. For a description of the `FInfo` structure, see the *Finder Interface Reference* .

`ioDirID`
: On input, the parent directory ID of the file. On output, the file's file ID.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlStBlk`
: On output, the first allocation block of the data fork.

`ioFlLgLen`
: On output, the logical size (the logical end-of-file) of the file's data fork, in bytes.

`ioFlPyLen`
: On output, the physical size (the physical end-of-file) of the file's data fork, in bytes.

`ioFlRStBlk`
: On output, the first allocation block of the resource fork.

`ioFlRLgLen`
: On output, the logical size of the file's resource fork, in bytes.

`ioFlRPyLen`
: On output, the physical size of the file's resource fork, in bytes.

`ioFlCrDat`
: On output, the date and time of the file's creation.

`ioFlMdDat`
: On output, the date and time of the file's last modification.

You should call `PBHGetFInfoAsync` just before `PBHSetFInfoAsync` (page 469) , so that the current information is present in the parameter block.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetFInfoSync

Obtains information about a file. (Deprecated in Mac OS X v10.4. Use `PBGetCatalogInfoSync` (page 139) instead.)

```
OSErr PBHGetFInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname. If the value of the `ioFDirIndex` field is negative or 0, `PBHGetFInfoSync` returns information about the file in the volume specified by the reference number in the `ioVRefNum` field and having the name given here. On output, a pointer to the name of the file, if the file is open. If you do not wish the name returned, pass `NULL` here.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file, or 0 for the default volume.

`ioFRefNum`
On output, the reference number of the first access path found, if the file is open and if the `ioFDirIndex` field is negative or 0; if the `ioFDirIndex` field is positive...

`ioFDirIndex`
On input, a directory index. If this value is positive, the function returns information about the file having the directory index specified here, on the volume specified in the `ioVRefNum` field and in the directory specified in the `ioDirID` field. If this value is negative or 0, the function returns information about the file on the specified volume, having the name pointed to in the `ioNamePtr` field.

`ioFlAttrib`
On output, the file attributes. See "File Attribute Constants" (page 300) for a description of the file attributes.

`ioFlFndrInfo`
On output, Finder information about the file. For a description of the `FInfo` data type, see the *Finder Interface Reference* .

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDirID`
: On input, the parent directory ID of the file. On output, the file's file ID.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlStBlk`
: On output, the first allocation block of the data fork.

`ioFlLgLen`
: On output, the logical size (the logical end-of-file) of the file's data fork, in bytes.

`ioFlPyLen`
: On output, the physical size (the physical end-of-file) of the file's data fork, in bytes.

`ioFlRStBlk`
: On output, the first allocation block of the resource fork.

`ioFlRLgLen`
: On output, the logical size of the resource fork, in bytes.

`ioFlRPyLen`
: On output, the physical size of the resource fork, in bytes.

`ioFlCrDat`
: On output, the date and time of the file's creation.

`ioFlMdDat`
: On output, the date and time of the file's last modification.

You should call `PBHGetFInfoSync` just before `PBHSetFInfoSync` (page 470), so that the current information is present in the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
ConvertMovieSndTrack
MovieBrowser
SoundPlayer

Declared In
`Files.h`

PBHGetLogInInfoAsync

Determines the login method used to log on to a particular shared volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHGetLogInInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the ObjParam (page 251) variant of the HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ioNamePtr

On input, a pointer to a pathname

ioVRefNum

On input, a volume specification for the shared volume. This field can contain a volume reference number, drive number, or 0 for the default volume.

ioObjType

On output, the login method type. See "Authentication Method Constants" (page 274) for the values that are recognized. Values in the range 7–127 are reserved for future use by Apple Computer, Inc. Values in the range 128–255 are available to your application as user-defined values.

ioObjNamePtr

On output, a pointer to the user name used to establish the session. The login user name is returned as a Pascal string. The maximum size of the user name is 31 characters.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBHGetLogInInfoSync

Determines the login method used to log on to a particular shared volume. (Deprecated in Mac OS X v10.4. There is no replacement function.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHGetLogInInfoSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHGetVInfoAsync

Gets detailed information about a volume. (Deprecated in Mac OS X v10.4. Use `PBGetVolumeInfoAsync` (page 146) instead.)

```
OSErr PBHGetVInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `HVolumeParam` (page 245) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to a buffer. If you specify a negative number in the `ioVolIndex` field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name. You should pass a pointer to a `Str31` value if you want the name returned. If you pass `NULL`, no volume name is returned.

`ioVRefNum`

On input, a volume specification for the volume for which to return information. If the `ioVolIndex` field is negative, the File Manager uses the value in the `ioNamePtr` field, along with the value specified in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVolIndex`

On input, an index used for indexing through all mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`

On output, the date and time of the volume's initialization.

`ioVLsMod`

On output, the date and time of the volume's last modification.

`ioVAtrb`

On output, the volume attributes. See "Volume Information Attribute Constants" (page 323) for a description of the volume attributes returned by this function.

`ioVNmFls`

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoAsync` (page 423) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`

On output, the first block of the volume bitmap.

`ioVAllocPtr`

On output, the block at which the search for the next new file allocation should start.

`ioVNmAlBlks`

On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`

On output, the size of the allocation blocks.

`ioVClpSiz`

On output, the default clump size.

`ioAlBlSt`

On output, the first block in the volume block map.

`ioVNxtCNID`

On output, the next unused catalog node ID.

`ioVFrBlk`

On output, the number of unused allocation blocks.

`ioVSigWord`

On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`

On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVDRefNum`
: On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
: On output, the file system handling this volume.

`ioVBkUp`
: On output, the date and time of the volume's last backup.

`ioVSeqNum`
: Used internally.

`ioVWrCnt`
: On output, the volume write count.

`ioVFilCnt`
: On output, the number of files on the volume.

`ioVDirCnt`
: On output, the number of directories on the volume.

`ioVFndrInfo`
: On output, Finder information for the volume.

You can get information about all the online volumes by making repeated calls to `PBHGetVInfoAsync`, starting with the value of the `ioVolIndex` field set to 1 and incrementing that value until `PBHGetVInfoAsync` returns `nsvErr`.

If you need to obtain information about HFS Plus volumes, you should use the `FSGetVolumeInfo` (page 74) function, or one of the corresponding parameter block calls, `PBGetVolumeInfoSync` (page 148) and `PBGetVolumeInfoAsync` (page 146). The `PBHGetVInfoAsync` function is still supported for HFS Plus volumes, but there is additional information returned by the `FSGetVolumeInfo` function (such as the date and time that the volume was last checked for consistency).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBHGetVInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

If the value of `ioVolIndex` is negative, the File Manager uses `ioNamePtr` and `ioVRefNum` in the standard way to determine the volume. However, because `PBHGetVInfoAsync` returns the volume name in the buffer whose address you passed in `ioNamePtr`, your input pathname will be modified. If you don't want your input pathname modified, make a copy of it and pass the copy to `PBHGetVInfoAsync`.

The volume name returned by `PBHGetVInfoAsync` is not a full pathname to the volume because it does not contain a colon.

For compatibility with older programs, some values returned by `PBHGetVInfoAsync` are not what is stored in the volume's Volume Control Block (VCB). Specifically:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

- ioVNmAlBlks and ioVFrBlk are pinned to values which, when multiplied by ioVAlBlkSiz, are always less than 2 Gigabytes.
- ioVNmAlBlks may not include the allocation blocks used by the catalog and extents overflow files.
- $4244 is returned in ioVSigWord for both HFS and HFS Plus volumes.

For unpinned total and free byte counts, and for the real ioVSigWord, use PBXGetVolInfoAsync (page 494) instead of PBHGetVInfoAsync.

Version Notes
In non-Carbon applications, you may pass a working directory reference in the ioVRefNum field; if you pass a working directory reference in that field, the number of files and directories in the specified directory is returned in the ioVNmFls field.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBHGetVInfoSync

Gets detailed information about a volume. (Deprecated in Mac OS X v10.4. Use PBGetVolumeInfoSync (page 148) instead.)

```
OSErr PBHGetVInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the HVolumeParam (page 245) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to a buffer. If you specify a negative number in the ioVolIndex field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name. You should pass a pointer to a Str31 value if you want the name returned. If you pass NULL, no volume name is returned.

ioVRefNum
On input, a volume reference number or drive number for the volume for which to return information; or 0 for the default volume. If the ioVolIndex field is negative, the File Manager uses the value in the ioNamePtr field, along with the value specified in the ioVRefNum field, to determine the volume. If the value in ioVolIndex is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVolIndex`

On input, an index used for indexing through all mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`

On output, the date and time of the volume's initialization.

`ioVLsMod`

On output, the date and time of the volume's last modification.

`ioVAtrb`

On output, the volume attributes. See "Volume Information Attribute Constants" (page 323) for a description of the volume attributes returned by this function.

`ioVNmFls`

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoSync` (page 426) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`

On output, the first block of the volume bitmap.

`ioVAllocPtr`

On output, the block at which the search for the next new file allocation should start.

`ioVNmAlBlks`

On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`

On output, the size of the allocation blocks.

`ioVClpSiz`

On output, the default clump size.

`ioAlBlSt`

On output, the first block in the volume block map.

`ioVNxtCNID`

On output, the next unused catalog node ID.

`ioVFrBlk`

On output, the number of unused allocation blocks.

`ioVSigWord`

On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`

On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

Mac OS X does not support drive numbers; in Mac OS X, the File Manager always returns a value of 1 in this field.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVDRefNum`
: On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
: On output, the file system handling this volume.

`ioVBkUp`
: On output, the date and time of the volume's last backup.

`ioVSeqNum`
: Used internally.

`ioVWrCnt`
: On output, the volume write count.

`ioVFilCnt`
: On output, the number of files on the volume.

`ioVDirCnt`
: On output, the number of directories on the volume.

`ioVFndrInfo`
: On output, Finder information for the volume.

You can get information about all the online volumes by making repeated calls to `PBHGetVInfoSync`, starting with the value of the `ioVolIndex` field set to 1 and incrementing that value until `PBHGetVInfoSync` returns `nsvErr`.

If you need to obtain information about HFS Plus volumes, you should use the `FSGetVolumeInfo` (page 74) function, or one of the corresponding parameter block calls, `PBGetVolumeInfoSync` (page 148) and `PBGetVolumeInfoAsync` (page 146). The `PBHGetVInfoSync` function is still supported for HFS Plus volumes, but there is additional information returned by the `FSGetVolumeInfo` function (such as the date and time that the volume was last checked for consistency).

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBHGetVInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

If the value of `ioVolIndex` is negative, the File Manager uses `ioNamePtr` and `ioVRefNum` in the standard way to determine the volume. However, because `PBHGetVInfoSync` returns the volume name in the buffer whose address you passed in `ioNamePtr`, your input pathname will be modified. If you don't want your input pathname modified, make a copy of it and pass the copy to `PBHGetVInfoSync`.

The volume name returned by `PBHGetVInfoSync` is not a full pathname to the volume because it does not contain a colon.

For compatibility with older programs, some values returned by `PBHGetVInfoSync` are not what is stored in the volume's Volume Control Block (VCB). Specifically:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

- `ioVNmAlBlks` and `ioVFrBlk` are pinned to values which, when multiplied by `ioVAlBlkSiz`, are always less than 2 Gigabytes.
- `ioVNmAlBlks` may not include the allocation blocks used by the catalog and extents overflow files.
- $4244 is returned in `ioVSigWord` for both HFS and HFS Plus volumes.

For unpinned total and free byte counts, and for the real `ioVSigWord`, use `PBXGetVolInfoSync` (page 497) instead of `PBHGetVInfoSync`.

Version Notes
In non-Carbon applications, you may pass a working directory reference in the `ioVRefNum` field; if you pass a working directory reference in that field, the number of files and directories in the specified directory is returned in the `ioVNmFls` field.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetVolAsync

Determines the default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetVolAsync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a working directory parameter block. See `WDPBRec` (page 263) for a description of the `WDPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBHGetVolAsync` function returns the default volume and directory last set by a call to `HSetVol` (page 373) or `PBHSetVolSync` (page 473). The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On output, a pointer to the default volume's name. You should pass a pointer to a `Str31` value if you want that name returned. If you pass `NULL` in this field, no volume name is returned.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
On output, the volume reference number of the default volume.

`ioWDProcID`
On output, the working directory user identifier.

`ioWDVRefNum`
On output, the volume reference number of the volume on which the default directory exists.

`ioWDDirID`
On output, the directory ID of the default directory.

Version Notes
When CarbonLib is not present, the `PBHGetVolAsync` function returns a working directory reference number in the `ioVRefNum` parameter if the previous call to `HSetVol` (page 373) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetVolSync

Determines the default volume and default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHGetVolSync (
   WDPBPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to a working directory parameter block. See `WDPBRec` (page 263) for a description of the `WDPBRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBHGetVolSync` function returns the default volume and directory last set by a call to `HSetVol` (page 373) or `PBHSetVolSync` (page 473). The relevant fields of the parameter block are:

`ioNamePtr`
On output, a pointer to the default volume's name. Pass a pointer to a `Str31` value if you want that name returned. If you pass `NULL` in this field, no volume name is returned.

`ioVRefNum`
On output, the volume reference number of the default volume.

`ioWDProcID`
On output, the working directory user identifier.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioWDVRefNum`
On output, the volume reference number of the volume on which the default directory exists.

`ioWDDirID`
On output, the directory ID of the default directory.

Version Notes
When CarbonLib is not present, the `PBHGetVolSync` function returns a working directory reference number in the `ioVRefNum` parameter if the previous call to `HSetVol` (page 373) (or one of the corresponding parameter block calls) passed in a working directory reference number.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHMoveRenameAsync

Moves a file or directory and optionally renames it. (Deprecated in Mac OS X v10.4. Use `FSMoveObjectAsync` (page 83) instead.)

```
OSErr PBHMoveRenameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to a `CopyParam` (page 191) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBHMoveRenameAsync` function allows you to move (not copy) a file or directory. The source and destination pathnames must point to the same file server volume. This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to the pathname for the source file or directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
: On input, a volume reference number or drive number for the volume containing the source file or directory. Pass 0 for the default volume.

`ioNewName`
: On input, a pointer to the destination pathname. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field. If `ioNewName` is not `NULL`, the destination directory is the directory having the partial pathname pointed to by `ioNewName` in the directory having ID `ioNewDirID` on the specified volume.

`ioCopyName`
: On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`
: On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`
: On input, the directory ID of the source directory.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHMoveRenameSync

Moves a file or directory and optionally renames it. (Deprecated in Mac OS X v10.4. Use `FSMoveObjectSync` (page 84) instead.)

```
OSErr PBHMoveRenameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a `CopyParam` (page 191) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBHMoveRenameSync` function allows you to move (not copy) a file or directory. The source and destination pathnames must point to the same file server volume. This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the pathname for the source file or directory.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the source file or directory. Pass 0 for the default volume.

`ioNewName`
On input, a pointer to the destination pathname. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field. If `ioNewName` is not `NULL`, the destination directory is the directory having the partial pathname pointed to by `ioNewName` in the directory having ID `ioNewDirID` on the specified volume.

`ioCopyName`
On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`
On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`
On input, the directory ID of the source directory.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHOpenAsync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 154) instead.)

```
OSErr PBHOpenAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic HFS parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. If you attempt to open a locked file for writing, `PBHOpenAsync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenAsync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

`ioNamePtr`
: On input, a pointer to the name of the file.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
: On output, a file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenAsync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`
: On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteAsync`, `PBSetEOFAsync` (page 483) , or `PBAllocateAsync` (page 373) call is made.

`ioDirID`
: On input, the directory ID of the file's parent directory.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If you use `PBHOpenAsync` to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use `PBHOpenDFAsync` (page 458) instead of `PBHOpenAsync`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDFAsync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 154) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHOpenDFAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HIOParam (page 241) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

You should use PBHOpenDFAsync instead of the PBHOpenAsync (page 457) function; PBHOpenDFAsync allows you to safely open a file whose name begins with a period (.).

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion function. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function. If you attempt to open a locked file for writing, PBHOpenDFAsync returns the result code permErr. If you request exclusive read/write permission but another access path is already open, PBHOpenDFAsync returns the reference number of the existing access path in ioRefNum and opWrErr as its function result.

ioNamePtr

On input, a pointer to the name of the file.

ioVRefNum

On input, a volume reference number, drive number, or 0 for the default volume.

ioRefNum

On output, the file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, PBHOpenDFAsync returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

ioPermssn

On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a PBWriteAsync, PBSetEOFAsync (page 483) , or PBAllocateAsync (page 373) call is made.

ioDirID

On input, the directory ID of the file's parent directory.

ioFVersNum

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the FSOpenFork (page 87) function, or one of its corresponding parameter block calls, PBOpenForkSync (page 155) or PBOpenForkAsync (page 154). If you try to open a fork larger than 2GB with the PBHOpenDFAsync function, you will receive an error message.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDFSync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 155) instead.)

```
OSErr PBHOpenDFSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). . If you attempt to open a locked file for writing, `PBHOpenDFSync` returns the result code `permErr`. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFSync` returns the reference number of the existing access path in `ioRefNum` and `opWrErr` as its function result.

Discussion
You should use `PBHOpenDFSync` instead of the `PBHOpenSync` (page 463) function; `PBHOpenDFSync` allows you to safely open a file whose name begins with a period (.).

The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, the file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, `PBHOpenDFSync` returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a `PBWriteSync`, `PBSetEOFSync` (page 484) , or `PBAllocateSync` (page 375) call is made.

`ioDirID`
On input, the directory ID of the file's parent directory.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `PBHOpenDFSync` function, you will receive an error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenRFAsync

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkAsync` (page 154) instead.)

```
OSErr PBHOpenRFAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
: A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. On some file systems, `PBHOpenRFAsync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

`ioNamePtr`
: On input, a pointer to the name of the file.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
: On output, a file reference number for accessing the open resource fork.

`ioPermssn`
: On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDirID`
: On input, the directory ID of the file's parent directory.

`ioFVersNum`
: On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `PBHOpenRFAsync` function, you will receive an error message.

Special Considerations

Generally your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `PBHOpenRFAsync` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/ write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenRFSync

Opens the resource fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 155) instead.)

```
OSErr PBHOpenRFSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). On some file systems, `PBHOpenRFSync` will return the error `eofErr` if you try to open the resource fork of a file for which no resource fork exists with read-only access.

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
On input, a pointer to the name of the file.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioRefNum`
On output, a file reference number for accessing the open resource fork.

`ioPermssn`
On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295).

`ioDirID`
On input, the directory ID of the file's parent directory.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Note that if you wish to access named forks other than the data and resource forks, or forks larger than 2GB, you will need to use the `FSOpenFork` (page 87) function, or one of its corresponding parameter block calls, `PBOpenForkSync` (page 155) or `PBOpenForkAsync` (page 154). If you try to open a fork larger than 2GB with the `PBOpenRFSync` function, you will receive an error message.

Special Considerations

Generally your application should use Resource Manager functions rather than File Manager functions to access a file's resource fork. The `PBHOpenRFSync` function does not read the resource map into memory and is generally useful only for applications (such as utilities that copy files) that need block-level access to a resource fork.

You should not use the resource fork of a file to hold non-resource data. Many parts of the system software assume that a resource fork always contains resource data.

Because there is no support for locking and unlocking file ranges on local disks in Mac OS X, regardless of whether File Sharing is enabled, you cannot open more than one path to a resource fork with read/ write permission. If you try to open a more than one path to a file's resource fork with `fsRdWrShPerm` permission, only the first attempt will succeed. Subsequent attempts will return an invalid reference number and the `ResError` function will return the error `opWrErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenSync

Opens the data fork of a file. (Deprecated in Mac OS X v10.4. Use `PBOpenForkSync` (page 155) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHOpenSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HIOParam (page 241) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you attempt to open a locked file for writing, PBHOpenSync returns the result code permErr. If you request exclusive read/write permission but another access path is already open, PBHOpenSync returns the reference number of the existing access path in ioRefNum and opWrErr as its function result.

Discussion

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to the name of the file.

ioVRefNum

On input, a volume reference number, drive number, or 0 for the default volume.

ioRefNum

On output, a file reference number for accessing the open data fork. If you request exclusive read/write permission but another access path is already open, PBHOpenSync returns the reference number of the existing access path. You should not use this reference number unless your application originally opened the file.

ioPermssn

On input, a constant specifying the type of access with which to open the fork. For a description of the types of access you can request, see "File Access Permission Constants" (page 295). You can open a path for writing even if it accesses a file on a locked volume, and no error is returned until a PBWriteSync, PBSetEOFSync (page 484) , or PBAllocateSync (page 375) call is made.

ioDirID

On input, the directory ID of the file's parent directory.

ioFVersNum

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If you use PBHOpenSync to try to open a file whose name begins with a period, you might mistakenly open a driver instead; subsequent attempts to write data might corrupt data on the target device. To avoid these problems, you should always use PBHOpenDFSync (page 460) instead of PBHOpenSync.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHRenameAsync

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `PBRenameUnicodeAsync` (page 161) instead.)

```
OSErr PBHRenameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the existing filename, directory name, or volume name.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioMisc`

On input, a pointer to the new name for the file, directory or volume.

`ioDirID`

On input, the parent directory ID of the file or directory to rename.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Given a pointer to the name of a file or directory in the `ioNamePtr` field, `PBHRenameAsync` changes it to the name pointed to in the `ioMisc` field. Given a pointer to a volume name in `ioNamePtr` or a volume reference number in `ioVRefNum`, the function changes the name of the volume to the name pointed to in `ioMisc`.

If a file ID reference exists for the file being renamed, the file ID remains with the file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 100) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 161) and `PBRenameUnicodeAsync` (page 161).

Special Considerations

You cannot use `PBHRenameAsync` to change the directory in which a file is located. To move a file or directory, use the `FSpCatMove` (page 348), `PBCatMoveSync` (page 380), or `PBCatMoveAsync` (page 379) functions.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHRenameSync

Renames a file, directory, or volume. (Deprecated in Mac OS X v10.4. Use `PBRenameUnicodeSync` (page 161) instead.)

```
OSErr PBHRenameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the existing filename, directory name, or volume name.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioMisc`
On input, a pointer to the new name for the file, directory or volume.

`ioDirID`
On input, the parent directory ID of the file or directory to rename.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

Given a pointer to the name of a file or directory in the `ioNamePtr` field, `PBHRenameSync` changes it to the name pointed to in the `ioMisc` field. Given a pointer to a volume name in `ioNamePtr` or a volume reference number in `ioVRefNum`, the function changes the name of the volume to the name pointed to in `ioMisc`.

If a file ID reference exists for the file being renamed, the file ID remains with the file.

To rename a file or directory using a long Unicode name, use the `FSRenameUnicode` (page 100) function or one of the corresponding parameter block calls, `PBRenameUnicodeSync` (page 161) and `PBRenameUnicodeAsync` (page 161).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

You cannot use `PBHRenameSync` to change the directory in which a file is located. To move a file or directory, use the `FSpCatMove` (page 348), `PBCatMoveSync` (page 380), or `PBCatMoveAsync` (page 379) functions.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHRstFLockAsync

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 162) instead.)

```
OSErr PBHRstFLockAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name for the file or directory o unlock.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`

On input, the parent directory ID of the file or directory to unlock.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `PBHRstFLockAsync` to unlock a directory. Otherwise, you can only use this function to unlock a file.

Access paths currently in use aren't affected by this function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHRstFLockSync

Unlocks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 163) instead.)

```
OSErr PBHRstFLockSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name for the file or directory to unlock.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDirID`
On input, the parent directory ID of the file or directory to unlock.

`ioFVersNum`
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function indicates that the volume supports folder locking (that is, the `bHasFolderLock` bit of the `vMAttrib` field is set), you can use `PBHRstFLockSync` to unlock a directory. Otherwise, you can only use this function to unlock a file.

Access paths currently in use aren't affected by this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHSetFInfoAsync

Sets information for a file. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 162) instead.)

```
OSErr PBHSetFInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the file.

`ioVRefNum`

On input, the volume reference number or drive number for the volume containing the file; or 0 for the default volume.

`ioFlFndrInfo`

On input, Finder information for the file. For a description of the `FInfo` data type, see the *Finder Interface Reference* .

`ioDirID`

On input, the parent directory ID for the file.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlCrDat`

On input, the date and time of the file's creation.

`ioFlMdDat`

On input, the date and time of the file's last modification.

You should call the `PBHGetFInfoAsync` (page 442) function just before calling `PBHSetFInfoAsync`, so that the current information is present in the parameter block.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHSetFInfoSync

Sets information for a file. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 163) instead.)

```
OSErr PBHSetFInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `HFileParam` (page 238) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the file.

`ioVRefNum`

On input, the volume reference number or drive number for the volume containing the file; or 0 for the default volume.

`ioFlFndrInfo`

On input, Finder information for the file. For a description of the `FInfo` data type, see the *Finder Interface Reference* .

`ioDirID`

On input, the parent directory ID of the file.

`ioFVersNum`

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

`ioFlCrDat`

On input, the date and time of the file's creation.

`ioFlMdDat`

On input, the date and time of the file's last modification.

You should call the `PBHGetFInfoSync` (page 444) function just before calling `PBHSetFInfoSync`, so that the current information is present in the parameter block.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHSetFLockAsync

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 162) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHSetFLockAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HFileParam (page 238) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

ioCompletion

On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult

On output, the result code of the function.

ioNamePtr

On input, a pointer to a name for the file or directory to lock.

ioVRefNum

On input, a volume reference number, drive number, or 0 for the default volume.

ioDirID

On input, the parent directory ID of the file or directory to lock.

ioFVersNum

On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the PBHGetVolParmsSync (page 518) or PBHGetVolParmsAsync (page 517) function indicates that the volume supports folder locking (that is, the bHasFolderLock bit of the vMAttrib field is set), you can use PBHSetFLockAsync to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. Access paths currently in use aren't affected.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

Files.h

PBHSetFLockSync

Locks a file or directory. (Deprecated in Mac OS X v10.4. Use PBSetCatalogInfoSync (page 163) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHSetFLockSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the HFileParam (page 238) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ioNamePtr
On input, a pointer to a name for the file or directory to lock.

ioVRefNum
On input, a volume reference number, drive number, or 0 for the default volume.

ioDirID
On input, the parent directory ID of the file or directory to lock.

ioFVersNum
On input, this field should be initialized to zero; if this field is not zero, the call will fall through to the now-obsolete Macintosh File System (MFS) code if the volume accessed is an MFS volume.

If the PBHGetVolParmsSync (page 518) or PBHGetVolParmsAsync (page 517) function indicates that the volume supports folder locking (that is, the bHasFolderLock bit of the vMAttrib field is set), you can use PBHSetFLockSync to lock a directory. Otherwise, you can only use this function to lock a file.

After you lock a file, all new access paths to that file are read-only. Access paths currently in use aren't affected.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
Files.h

PBHSetVolAsync

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHSetVolAsync (
   WDPBPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a working directory parameter block. See WDPBRec (page 263) for a description of the WDPBRec data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname. If this field specifies a full pathname, the default volume is set to the volume whose name is contained in that pathname. (A full pathname overrides the `ioVRefNum` field.)If this field contains a partial pathname and the `ioVRefNum` field specifies a volume reference number, then the default directory is set to the directory having the partial pathname specified here, in the directory given in the `ioWDDirID` field. If this field is `NULL`, then the default directory is set to the directory having the ID specified in the `ioWDDirID` field.

`ioVRefNum`
: On input, a volume reference number for the default volume. This field is ignored if the `ioNamePtr` field specifies a full pathname.

`ioWDDirID`
: On input, a directory ID. If the `ioVRefNum` field contains a volume reference number and `ioNamePtr` contains a partial pathname, this field contains the directory ID of the directory containing the default directory. If `ioNamePtr` is `NULL`, this field contains the directory ID of the default directory.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 0.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetVolSync

Sets the default volume and the default directory. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBHSetVolSync (
   WDPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a working directory parameter block. See `WDPBRec` (page 263) for a description of the `WDPBRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to a pathname. If this field specifies a full pathname, the default volume is set to the volume whose name is contained in that pathname. (A full pathname overrides the `ioVRefNum` field.)If this field contains a partial pathname and the `ioVRefNum` field specifies a volume reference number, then the default directory is set to the directory having the partial pathname specified here, in the directory given in the `ioWDDirID` field. If this field is `NULL`, then the default directory is set to the directory having the ID specified in the `ioWDDirID` field.

`ioVRefNum`
: On input, the volume reference number for the default volume. This field is ignored if the `ioNamePtr` field specifies a full pathname.

`ioWDDirID`
: On input, a directory ID. If the `ioVRefNum` field contains a volume reference number and `ioNamePtr` contains a partial pathname, this field contains the directory ID of the directory containing the default directory. If `ioNamePtr` is `NULL`, this field contains the directory ID of the default directory.

Both the default volume and the default directory are used in calls made with no volume name, a volume reference number of 0, and a directory ID of 0.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBLockRangeAsync

Locks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXLockRangeAsync` (page 172) instead.)

```
OSErr PBLockRangeAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. If you call `PBLockRangeAsync` on a file system that does not implement it—for example, SMB—`PBLockRangeAsync` returns `noErr` and does nothing.

`ioRefNum`
: On input, the file reference number of the file owning the range to lock.

`ioReqCount`
: On input, the number of bytes in the range. Set `ioReqCount` to –1 to lock the maximum number of bytes from the position specified in the `ioPosOffset` field.

`ioPosMode`
: On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 314) for more information on the constants you can use to specify the base location.

  You should not use the `fsFromLEOF` constant when locking a file range. `PBLockRangeAsync` does not return the start of the locked range; thus, there is no way to determine what range was actually locked when you specify `fsFromLEOF`.

`ioPosOffset`
: On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBLockRangeAsync` function locks a portion of a file that was opened with shared read/write permission. The beginning of the range to be locked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be locked is determined by the beginning of the range and the `ioReqCount` field. For example, to lock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0.

The `PBLockRangeAsync` function uses the same parameters as both `PBReadAsync` and `PBWriteAsync`; by calling it immediately before `PBReadAsync`, you can use the information in the parameter block for the `PBReadAsync` call.

When you're finished with the data (typically after a call to `PBWriteSync`), you can call `PBUnlockRangeAsync` (page 490) to free that portion of the file for subsequent read and write calls. Closing a file also releases all locked ranges in that file.

Special Considerations

The `PBLockRangeAsync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a sharable local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 233) returned by the `PBHGetVolParmsSync` (page 518) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBLockRangeSync

Locks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXLockRangeSync` (page 173) or `FSLockRange` (page 78) instead.)

```
OSErr PBLockRangeSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). If you call `PBLockRangeSync` on a file system that does not implement it—for example, SMB—`PBLockRangeSync` returns `noErr` and does nothing.

Discussion

The relevant fields of the parameter block are:

`ioRefNum`

On input, the file reference number of the file owning the range to lock.

`ioReqCount`

On input, the number of bytes in the range. Set `ioReqCount` to –1 to lock the maximum number of bytes from the position specified in the `ioPosOffset` field.

`ioPosMode`

On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 314) for more information about the constants you can use to specify the base location.

You should not use the `fsFromLEOF` constant when locking a file range. `PBLockRangeSync` does not return the start of the locked range; thus, there is no way to determine what range was actually locked when you specify `fsFromLEOF`.

`ioPosOffset`

On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBLockRangeSync` function locks a portion of a file that was opened with shared read/write permission. The beginning of the range to be locked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be locked is determined by the beginning of the range and the `ioReqCount` field. For example, to lock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0.

The `PBLockRangeSync` function uses the same parameters as both `PBReadSync` and `PBWriteSync`; by calling it immediately before `PBReadSync`, you can use the information in the parameter block for the `PBReadSync` call.

When you're finished with the data (typically after a call to `PBWriteSync`), you can call `PBUnlockRangeSync` (page 491) to free that portion of the file for subsequent read and write calls. Closing a file also releases all locked ranges in that file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

The `PBLockRangeSync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a sharable local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 233) returned by the `PBHGetVolParmsSync` (page 518) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBMakeFSSpecAsync

Creates an `FSSpec` structure for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBMakeFSRefUnicodeAsync` (page 151) instead.)

```
OSErr PBMakeFSSpecAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `PBMakeFSSpecAsync` fills in the structure and returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

`PBMakeFSSpecAsync` can return a number of different File Manager error codes. When `PBMakeFSSpecAsync` returns any result other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0.

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function. When `PBMakeFSSpecAsync` returns any result other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0. See "File Manager Result Codes".

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`

On input, a pointer to a full or partial pathname specifying the file or directory for which to create an `FSSpec`. If the `ioNamePtr` field specifies a full pathname, `PBMakeFSSpecAsync` ignores both the `ioVRefNum` and `ioDirID` fields. A partial pathname might identify only the final target, or it might include one or more parent directory names. If `ioNamePtr` specifies a partial pathname, then `ioVRefNum`, `ioDirID`, or both must be valid.

`ioVRefNum`

On input, a volume specification for the volume containing the file or directory. This field can contain a volume reference number, a drive number, or 0 to specify the default volume.

`ioMisc`

On input, a pointer to an `FSSpec` (page 226) structure. Given a complete specification for a file or directory, the `PBMakeFSSpecAsync` function fills in this `FSSpec` structure to identify the file or directory. On output, this field points to the initialized `FSSpec`. The file system specification structure that you pass in this field should not share storage space with the input pathname; the `name` field may be initialized to the empty string before the pathname has been processed. For example, `ioNamePtr` should not refer to the `name` field of the output file system specification.

`ioDirID`

On input, a directory ID. This field usually specifies the parent directory ID of the target object. If the directory is sufficiently specified by the `ioNamePtr` field, the `ioDirID` field can be set to 0. If the `ioNamePtr` field contains an empty string, `PBMakeFSSpecAsync` creates an `FSSpec` structure for the directory specified by the `ioDirID` field.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `PBMakeFSSpecAsync` fills in the structure and returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the `ioVRefNum` field. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBMakeFSSpecSync

Creates an `FSSpec` structure for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBMakeFSRefUnicodeSync` (page 151) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBMakeFSSpecSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). When `PBMakeFSSpecSync` returns any result other than `noErr` or `fnfErr`, all fields of the resulting `FSSpec` structure are set to 0.

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a full or partial pathname specifying the file or directory for which to create an `FSSpec`. If the `ioNamePtr` field specifies a full pathname, `PBMakeFSSpecSync` ignores both the `ioVRefNum` and `ioDirID` fields. A partial pathname might identify only the final target, or it might include one or more parent directory names. If `ioNamePtr` specifies a partial pathname, then `ioVRefNum`, `ioDirID`, or both must be valid.

`ioVRefNum`

On input, a volume specification for the volume containing the file or directory. This field can contain a volume reference number, a drive number, or 0 to specify the default volume.

`ioMisc`

On input, a pointer to an `FSSpec` (page 226) structure. Given a complete specification for a file or directory, the `PBMakeFSSpecSync` function fills in this `FSSpec` structure to identify the file or directory. On output, this field points to the initialized `FSSpec`. The file system specification structure that you pass in this field should not share storage space with the input pathname; the `name` field may be initialized to the empty string before the pathname has been processed. For example, `ioNamePtr` should not refer to the `name` field of the output file system specification.

`ioDirID`

On input, a directory ID. This field usually specifies the parent directory ID of the target object. If the directory is sufficiently specified by the `ioNamePtr` field, the `ioDirID` field can be set to 0. If the `ioNamePtr` field contains an empty string, `PBMakeFSSpecSync` creates an `FSSpec` structure for the directory specified by the `ioDirID` field.

If the specified volume is mounted and the specified parent directory exists, but the target file or directory doesn't exist in that location, `PBMakeFSSpecSync` fills in the structure and returns `fnfErr` instead of `noErr`. The structure is valid, but it describes a target that doesn't exist. You can use the structure for another operation, such as creating a file.

Carbon Porting Notes

Non-Carbon applications can also specify a working directory reference number in the `ioVRefNum` field. However, because working directories are not supported in Carbon, you cannot specify a working directory reference number if you wish your application to be Carbon-compatible.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
`Files.h`

PBSetCatInfoAsync

Modifies catalog information for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoAsync` (page 162) instead.)

```
OSErr PBSetCatInfoAsync (
   CInfoPBPtr paramBlock
);
```

Parameters

paramBlock
A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 187) for a description of the `CInfoPBRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The `PBSetCatInfoAsync` function sets information about a file or directory. When used to set information about a file, it works much as `PBHSetFInfoAsync` (page 469) does, but lets you set some additional information.

If the object is a file, the relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFlFndrInfo`
On input, Finder information for the file.

`ioDirID`
On input, the parent directory ID of the file.

`ioFlCrDat`
On input, the date and time of the file's creation.

`ioFlMdDat`
On input, the date and time of the file's last modification.

`ioFlBkDat`
On input, the date and time of the file's last backup.

`ioFlXFndrInfo`
On input, extended Finder information.

If the object is a directory, the relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname.

`ioVRefNum`
: On input, a volume reference number, drive number, or 0 for the default volume.

`ioDrUsrWds`
: On input, information used by the Finder.

`ioDrDirID`
: On input, the directory ID.

`ioDrCrDat`
: On input, the date and time of the directory's creation.

`ioDrMdDat`
: On input, the date and time of the directory's last modification.

`ioDrBkDat`
: On input, the date and time of the directory's last backup.

`ioDrFndrInfo`
: On input, additional information used by the Finder.

To modify the catalog information for a named fork other than the data and resource fork, or to modify other catalog information maintained on HFS Plus volumes that is not modifiable through `PBSetCatInfoAsync`, use one of the functions, `FSSetCatalogInfo` (page 101) , `PBSetCatalogInfoSync` (page 163) , or `PBSetCatalogInfoAsync` (page 162).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetCatInfoSync

Modifies catalog information for a file or directory. (Deprecated in Mac OS X v10.4. Use `PBSetCatalogInfoSync` (page 163) instead.)

```
OSErr PBSetCatInfoSync (
   CInfoPBPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to an HFS catalog information parameter block. See `CInfoPBRec` (page 187) for a description of the `CInfoPBRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The `PBSetCatInfoSync` function sets information about a file or directory. When used to set information about a file, it works much as `PBHSetFInfoSync` (page 470) does, but lets you set some additional information.

If the object is a file, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioFlFndrInfo`
On input, Finder information for the file.

`ioDirID`
On input, the parent directory ID of the file.

`ioFlCrDat`
On input, the date and time of the file's creation.

`ioFlMdDat`
On input, the date and time of the file's last modification.

`ioFlBkDat`
On input, the date and time of the file's last backup.

`ioFlXFndrInfo`
On input, extended Finder information.

If the object is a directory, the relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume reference number, drive number, or 0 for the default volume.

`ioDrUsrWds`
On input, information used by the Finder.

`ioDrDirID`
On input, the directory ID.

`ioDrCrDat`
On input, the date and time of the directory's creation.

`ioDrMdDat`
On input, the date and time of the directory's last modification.

`ioDrBkDat`
On input, the date and time of the directory's last backup.

`ioDrFndrInfo`
On input, additional information used by the Finder.

To modify the catalog information for a named fork other than the data and resource fork, or to modify other catalog information maintained on HFS Plus volumes that is not modifiable through `PBSetCatInfoSync`, use one of the functions, `FSSetCatalogInfo` (page 101) , `PBSetCatalogInfoSync` (page 163) , or `PBSetCatalogInfoAsync` (page 162).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetEOFAsync

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBSetForkSizeAsync` (page 166) instead.)

```
OSErr PBSetEOFAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for the open file.

`ioMisc`
On input, the new logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you must coerce the desired value from a long integer to type `Ptr`. If the value of the `ioMisc` field is 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

If you attempt to set the logical end-of-file beyond the current physical end-of-file, another allocation block is added to the file if there isn't enough space on the volume, no change is made and `PBSetEOFAsync` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 502) , `PBFlushVolSync` (page 510) , or `PBFlushVolAsync` (page 509). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 167) and `PBSetForkSizeAsync` (page 166).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetEOFSync

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `PBSetForkSizeSync` (page 167) instead.)

```
OSErr PBSetEOFSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`

On input, a file reference number for the open file.

`ioMisc`

On input, the new logical size (the logical end-of-file) of the given file. Because the `ioMisc` field is of type `Ptr`, you must coerce the desired value from a long integer to type `Ptr`. If the value of the `ioMisc` field is 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

If you attempt to set the logical end-of-file beyond the current physical end-of-file, another allocation block is added to the file if there isn't enough space on the volume, no change is made and `PBSetEOFSync` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 502) , `PBFlushVolSync` (page 510) , or `PBFlushVolAsync` (page 509). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 167) and `PBSetForkSizeAsync` (page 166).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBSetForeignPrivsAsync

Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBSetForeignPrivsAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBSetForeignPrivsSync

Changes the native access-control information for a file or directory stored on a volume managed by a foreign file system. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBSetForeignPrivsSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBSetFPosAsync

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `PBSetForkPositionAsync` (page 165) instead.)

```
OSErr PBSetFPosAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioRefNum`
: On input, the file reference number for an open file.

`ioPosMode`
: On input, a constant indicating how to position the mark; this field must contain one of the values described in "Position Mode Constants" (page 314).

`ioPosOffset`
: On input, the offset from the base location specified by the `ioPosMode` field for the file mark. If you specify `fsAtMark` in the `ioPosMode` field, the mark is left wherever it's currently positioned and the value in the `ioPosOffset` field is ignored. If you specify `fsFromLEOF`, the value in `ioPosOffset` must be less than or equal to 0. On output, the position at which the mark was actually set.

The `PBSetFPosAsync` function sets the mark of the specified file to the position specified by the `ioPosMode` and `ioPosOffset` fields. If you try to set the mark past the logical end-of-file, `PBSetFPosAsync` moves the mark to the end-of-file and returns `eofErr` as its function result.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 165) and `PBSetForkPositionAsync` (page 165).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetFPosSync

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `PBSetForkPositionSync` (page 165) instead.)

```
OSErr PBSetFPosSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
: A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
On input, the file reference number for an open file.

`ioPosMode`
On input, a constant indicating how to position the file mark; this field must contain one of the values described in "Position Mode Constants" (page 314).

`ioPosOffset`
On input, the offset from the base location specified by the `ioPosMode` field for the file mark. If you specify `fsAtMark` in the `ioPosMode` field, the mark is left wherever it's currently positioned and the value in the `ioPosOffset` field is ignored. If you specify `fsFromLEOF`, the value in `ioPosOffset` must be less than or equal to 0. On output, the position at which the mark was actually set.

The `PBSetFPosSync` function sets the mark of the specified file to the position specified by the `ioPosMode` and `ioPosOffset` fields. If you try to set the mark past the logical end-of-file, `PBSetFPosSync` moves the mark to the end-of-file and returns `eofErr` as its function result.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 165) and `PBSetForkPositionAsync` (page 165).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetVInfoAsync

Changes information about a volume. (Deprecated in Mac OS X v10.4. Use `PBSetVolumeInfoAsync` (page 168) instead.)

```
OSErr PBSetVInfoAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HVolumeParam` (page 245) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to the volume's name. You can specify a new name for the volume here. You cannot specify the volume by name you must use either the volume reference number or the drive number.

`ioVRefNum`
: On input, a volume reference number or drive number for the volume whose information is to be changed; or 0 for the default volume.

`ioVCrDate`
: On input, the date and time of the volume's initialization.

`ioVLsMod`
: On input, the date and time of the volume's last modification.

`ioVAtrb`
: On input, the volume attributes. Only bit 15 of the `ioVAtrb` field can be changed; setting it locks the volume. See "Volume Information Attribute Constants" (page 323) for a description of the volume attributes.

`ioVBkUp`
: On input, the date and time of the volume's last backup.

`ioVSeqNum`
: Used internally.

`ioVFndrInfo`
: On input, Finder information for the volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBSetVInfoSync

Changes information about a volume. (Deprecated in Mac OS X v10.4. Use `PBSetVolumeInfoSync` (page 169) instead.)

```
OSErr PBSetVInfoSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HVolumeParam` (page 245) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the volume's name. You can specify a new name for the volume here. You cannot specify the volume by name you must use either the volume reference number or the drive number.

`ioVRefNum`
On input, a volume reference number or drive number for the volume whose information is to be changed; or 0 for the default volume.

`ioVCrDate`
On input, the date and time of the volume's initialization.

`ioVLsMod`
On input, the date and time of the volume's last modification.

`ioVAtrb`
On input, the volume attributes. Only bit 15 of the `ioVAtrb` field can be changed; setting it locks the volume. See "Volume Information Attribute Constants" (page 323) for a description of the volume attributes.

`ioVBkUp`
On input, the date and time of the volume's last backup.

`ioVSeqNum`
Used internally.

`ioVFndrInfo`
On input, Finder information for the volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBShareAsync

Establishes a local volume or directory as a share point. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBShareAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBShareSync

Establishes a local volume or directory as a share point. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBShareSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBUnlockRangeAsync

Unlocks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXUnlockRangeAsync` (page 173) instead.)

```
OSErr PBUnlockRangeAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. If you call `PBUnlockRangeAsync` on a file system that does not implement it—for example, SMB—`PBUnlockRangeAsync` returns `noErr` and does nothing.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioRefNum`
: On input, the file reference number of the file owning the range to unlock.

`ioReqCount`
: On input, the number of bytes in the range.

`ioPosMode`
: On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 314) for more information on the constants you can use to indicate the base location.

`ioPosOffset`
: On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBUnlockRangeAsync` function unlocks a portion of a file that you locked with `PBLockRangeSync` (page 476) or `PBLockRangeAsync` (page 474). The beginning of the range to be unlocked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be unlocked is determined by the beginning of the range and the `ioReqCount` field. For example, to unlock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0. The range of bytes to be unlocked must be the exact same range locked by a previous call to `PBLockRangeSync` (page 476) or `PBLockRangeAsync` (page 474).

If for some reason you need to unlock a range whose beginning or length is unknown, you can simply close the file. When a file is closed, all locked ranges held by the user are unlocked.

Special Considerations

The `PBUnlockRangeAsync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 233) returned by the `PBHGetVolParmsSync` (page 518) function is set.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBUnlockRangeSync

Unlocks a portion of a file. (Deprecated in Mac OS X v10.4. Use `PBXUnlockRangeSync` (page 173) or `FSUnlockRange` (page 104) instead.)

```
OSErr PBUnlockRangeSync (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329). If you call `PBUnlockRangeSync` on a file system that does not implement it—for example, SMB—`PBUnlockRangeSync` returns `noErr` and does nothing.

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
On input, the file reference number of the file owning the range to unlock.

`ioReqCount`
On input, the number of bytes in the range.

`ioPosMode`
On input, a constant specifying the base location for the start of the locked range. See "Position Mode Constants" (page 314) for more information on the constants you can use to indicate the base location.

`ioPosOffset`
On input, the offset from the base location specified in the `ioPosMode` field for the start of the locked range.

The `PBUnlockRangeSync` function unlocks a portion of a file that you locked with `PBLockRangeSync` (page 476) or `PBLockRangeAsync` (page 474). The beginning of the range to be unlocked is determined by the `ioPosMode` and `ioPosOffset` fields. The end of the range to be unlocked is determined by the beginning of the range and the `ioReqCount` field. For example, to unlock the first 50 bytes in a file, set `ioReqCount` to 50, `ioPosMode` to `fsFromStart`, and `ioPosOffset` to 0. The range of bytes to be unlocked must be the exact same range locked by a previous call to `PBLockRangeSync` (page 476) or `PBLockRangeAsync` (page 474).

If for some reason you need to unlock a range whose beginning or length is unknown, you can simply close the file. When a file is closed, all locked ranges held by the user are unlocked.

Special Considerations

The `PBUnlockRangeSync` function does nothing if the file specified in the `ioRefNum` field is open with shared read/write permission but is not located on a remote server volume or is not located under a share point on a local volume. To check whether file sharing is currently on, check that the `bHasPersonalAccessPrivileges` bit in the `vMAttrib` field of the `GetVolParmsInfoBuffer` (page 233) returned by the `PBHGetVolParmsSync` (page 518) function is set.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBUnmountVol

Unmounts a volume. (Deprecated in Mac OS X v10.4. Use `FSEjectVolumeSync` (page 59) or `FSUnmountVolumeSync` (page 105) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBUnmountVol (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `VolumeParam` (page 259) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the volume.

`ioVRefNum`

On input, the volume reference number of the volume to unmount, or 0 for the default volume.

This function calls `PBFlushVolSync` to flush the specified volume, unmounts and ejects the volume, and releases the memory used for the volume. Prior to calling this function, all user files on the volume must be closed. Ejecting a volume results in the unmounting of other volumes on the same device.

The `PBUnmountVol` function always executes synchronously.

Special Considerations

Don't unmount the startup volume. Doing so will cause a system crash.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Files.h`

PBUnshareAsync

Makes a share point unavailable on the network. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBUnshareAsync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Not available to 64-bit applications.

Declared In
`Files.h`

PBUnshareSync

Makes a share point unavailable on the network. (Deprecated in Mac OS X v10.4. There is no replacement function.)

```
OSErr PBUnshareSync (
   HParmBlkPtr paramBlock
);
```

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBXGetVolInfoAsync

Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated in Mac OS X v10.4. Use `FSGetVolumeInfo` (page 74) instead.)

```
OSErr PBXGetVolInfoAsync (
   XVolumeParamPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to an extended volume parameter block. See `XVolumeParam` (page 268) for a description of the `XVolumeParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the function result.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`

On input, a pointer to a buffer. You should pass a pointer to a `Str31` value if you want the volume name returned; otherwise, pass `NULL`. If you specify a negative number in the `ioVolIndex` field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume. If the value in the `ioVolIndex` field is negative, the File Manager uses the name in the `ioNamePtr` field, along with the value in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

`ioXVersion`

On input, the version of the extended volume parameter block. Currently, this value is 0.

`ioVolIndex`

On input, an index used for indexing through all the mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

`ioVCrDate`

On output, the date and time of the volume's creation (initialization).

`ioVLsMod`

On output, the date and time that the volume was last modified.

`ioVAtrb`

On output, the volume attributes. See "Volume Information Attribute Constants" (page 323) for a description of these attributes.

`ioVNmFls`

On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoAsync` (page 423) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`

On output, the first block of the volume bitmap.

`ioVAllocPtr`

On output, the block where the next new file allocation search should start.

`ioVNmAlBlks`

On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`

On output, the allocation block size for the volume.

`ioVClpSiz`

On output, the volume's default clump size.

`ioAlBlSt`

On output, the first block in the volume block map.

`ioVNxtCNID`

On output, the next unused catalog node ID.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVFrBlk`
: On output, the number of free (unused) allocation blocks on the volume.

`ioVSigWord`
: On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
: On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

`ioVDRefNum`
: On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
: On output, the file system ID for the file system handling this volume.

`ioVBkUp`
: On output, the date and time that the volume was last backed up.

`ioVSeqNum`
: Used internally.

`ioVWrCnt`
: On output, the volume write count.

`ioVFilCnt`
: On output, the number of files on the volume.

`ioVDirCnt`
: On output, the number of directories on the volume.

`ioVFndrInfo`
: On output, Finder information for the volume.

`ioVTotalBytes`
: On output, the total number of bytes on the volume.

`ioVFreeBytes`
: On output, the number of free bytes on the volume.

The `PBXGetVolInfoAsync` function is similar to the `PBHGetVInfoAsync` (page 447) function except that it returns additional volume space information in 64-bit integers and does not modify the information copied from the volume's volume control block (VCB). Systems that support `PBXGetVolInfoAsync` will have the `gestaltFSSupports2TBVols` bit set in the response returned by the `gestaltFSAttr Gestalt` selector. See *Inside Mac OS X: Gestalt Manager Reference* for a description of the `gestaltFSAttr` selector and of the bits that may be returned in the response.

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBXGetVolInfoAsync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

PBXGetVolInfoSync

Returns information about a volume, including size information for volumes up to 2 terabytes. (Deprecated in Mac OS X v10.4. Use `FSGetVolumeInfo` (page 74) instead.)

```
OSErr PBXGetVolInfoSync (
   XVolumeParamPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to an extended volume parameter block. See `XVolumeParam` (page 268) for a description of the `XVolumeParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a buffer. You should pass a pointer to a `Str31` value if you want the volume name returned; otherwise, pass `NULL`. If you specify a negative number in the `ioVolIndex` field, this buffer should hold the name of the volume for which to return information. On output, a pointer to the volume's name.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume. If the value in the `ioVolIndex` field is negative, the File Manager uses the name in the `ioNamePtr` field, along with the value in the `ioVRefNum` field, to determine the volume. If the value in `ioVolIndex` is 0, the File Manager attempts to access the volume using only the value in this field. On output, the volume reference number.

`ioXVersion`

On input, the version of the extended volume parameter block. Currently, this value is 0.

`ioVolIndex`

On input, an index used for indexing through all the mounted volumes. If this value is positive, the File Manager uses it to find the volume for which to return information. For instance, if the value of `ioVolIndex` is 2, the File Manager attempts to access the second mounted volume in the VCB queue. If `ioVolIndex` is negative, the File Manager uses the values in the `ioNamePtr` and `ioVRefNum` fields to access the requested volume. If `ioVolIndex` is 0, the File Manager uses only the value in the `ioVRefNum` field.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVCrDate`
: On output, the date and time of the volume's creation (initialization).

`ioVLsMod`
: On output, the date and time that the volume was last modified.

`ioVAtrb`
: On output, the volume attributes. See "Volume Information Attribute Constants" (page 323) for a description of these attributes.

`ioVNmFls`
: On output, the number of files in the root directory of the volume. For performance reasons, the Carbon File Manager does not return the number of files in this field; instead, it sets `ioVNmFls` to 0.To determine the number of files in the root directory of a volume in Carbon, call `PBGetCatInfoSync` (page 426) for the root directory. The number of files in the root directory is returned in the `ioDrNmFls` field.

`ioVBitMap`
: On output, the first block of the volume bitmap.

`ioVAllocPtr`
: On output, the block where the next new file allocation search should start.

`ioVNmAlBlks`
: On output, the number of allocation blocks on the volume.

`ioVAlBlkSiz`
: On output, the allocation block size for the volume.

`ioVClpSiz`
: On output, the volume's default clump size.

`ioAlBlSt`
: On output, the first block in the volume block map.

`ioVNxtCNID`
: On output, the next unused catalog node ID.

`ioVFrBlk`
: On output, the number of free (unused) allocation blocks on the volume.

`ioVSigWord`
: On output, the volume signature. For HFS volumes, this is 'BD' for HFS Plus volumes, this is 'H+'.

`ioVDrvInfo`
: On output, the drive number. You can determine whether the given volume is online by inspecting the value of this field. For online volumes, the `ioVDrvInfo` field contains the drive number of the drive containing the specified volume and hence is always greater than 0. If the value returned in `ioVDrvInfo` is 0, the volume is either offline or ejected.

`ioVDRefNum`
: On output, the driver reference number. You can determine whether the volume is offline or ejected by inspecting the value of this field. If the volume is offline, the value of `ioVDRefNum` is the negative of the drive number (which is cleared when the volume is placed offline; hence the `ioVDrvInfo` field for an offline volume is zero), and is a negative number. If the volume is ejected, the value of `ioVDRefNum` is the drive number itself, and thus is a positive number. For online volumes, `ioVDRefNum` contains a driver reference number; these numbers are always less than 0.

`ioVFSID`
: On output, the file system ID for the file system handling this volume.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVBkUp`
On output, the date and time that the volume was last backed up.

`ioVSeqNum`
Used internally.

`ioVWrCnt`
On output, the volume write count.

`ioVFilCnt`
On output, the number of files on the volume.

`ioVDirCnt`
On output, the number of directories on the volume.

`ioVFndrInfo`
On output, Finder information for the volume.

`ioVTotalBytes`
On output, the total number of bytes on the volume.

`ioVFreeBytes`
On output, the number of free bytes on the volume.

The `PBXGetVolInfoSync` function is similar to the `PBHGetVInfoSync` (page 450) function except that it returns additional volume space information in 64-bit integers and does not modify the information copied from the volume's volume control block (VCB). Systems that support `PBXGetVolInfoSync` will have the `gestaltFSSupports2TBVols` bit set in the response returned by the `gestaltFSAttr Gestalt` selector. See *Inside Mac OS X: Gestalt Manager Reference* for a description of the `gestaltFSAttr` selector and of the bits that may be returned in the response.

Special Considerations

After an operation that changes the amount of free space on the volume—such as deleting a file—there may be a delay before a call to `PBXGetVolInfoSync` returns the updated amount. This is because the File Manager caches and periodically updates file system information, to reduce the number of calls made to retrieve the information from the file system. Currently, the File Manager updates its information every 15 seconds. This primarily affects NFS volumes. DOS, SMB, UFS and WebDAV volumes were also affected by this in previous versions of Mac OS X, but behave correctly in Mac OS X version 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

SetEOF

Sets the logical size of an open file. (Deprecated in Mac OS X v10.4. Use `FSSetForkSize` (page 102) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr SetEOF (
   FSIORefNum refNum,
   SInt32 logEOF
);
```

Parameters

refNum

The file reference number of an open file.

logEOF

The new logical size (the logical end-of-file) of the given file. If you set the `logEOF` parameter to 0, all space occupied by the file on the volume is released. The file still exists, but it contains 0 bytes. Setting a file fork's end-of-file to 0 is therefore not the same as deleting the file, which removes both file forks at once.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

If you attempt to set the logical end-of-file beyond the physical end-of-file, the physical end-of-file is set 1 byte beyond the end of the next free allocation block if there isn't enough space on the volume, no change is made, and `SetEOF` returns `dskFulErr` as its function result.

To ensure that your changes to the file are written to disk, call one of the functions, `FlushVol` (page 502) , `PBFlushVolSync` (page 510) , or `PBFlushVolAsync` (page 509). To set the size of a named fork other than the data and resource forks, or to grow the size of a file beyond 2GB, you must use the `FSSetForkSize` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkSizeSync` (page 167) and `PBSetForkSizeAsync` (page 166).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

qtcompress

qtcompress.win

qtdataexchange

qtdataexchange.win

qtinfo

Declared In

`Files.h`

SetFPos

Sets the position of the file mark. (Deprecated in Mac OS X v10.4. Use `FSSetForkPosition` (page 102) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr SetFPos (
   FSIORefNum refNum,
   SInt16 posMode,
   SInt32 posOff
);
```

Parameters

refNum

The file reference number of an open file.

posMode

A constant specifying how to position the file mark; this parameter must contain one of the values described in "Position Mode Constants" (page 314).

posOff

The offset from the base location specified by the `posMode` parameter for the new file mark position. If you specify `fsFromLEOF` in the `posMode` parameter, the value in the `posOff` parameter must be less than or equal to 0. If you specify `fsAtMark`, the value in the `posOff` parameter is ignored.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

Because the read and write operations performed by the functions `FSRead` (page 359) and `FSWrite` (page 360) begin at the current mark, you may want to call `SetFPos` to reposition the file mark before reading from or writing to the file.

To set the file mark position for a named fork other than the data and resource forks, or to position the file mark at a point more than 2GB into the file, use the `FSSetForkPosition` (page 102) function, or one of the corresponding parameter block calls, `PBSetForkPositionSync` (page 165) and `PBSetForkPositionAsync` (page 165).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

JPEG File Interchange Format

qtdataexchange

qtgraphimp

qtmoviefromprocs

qtmoviefromprocs.win

Declared In

`Files.h`

UnmountVol

Unmounts a volume that isn't currently being used. (Deprecated in Mac OS X v10.4. Use `FSUnmountVolumeSync` (page 105) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr UnmountVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum
);
```

Parameters

volName

The name of a mounted volume. This parameter may be `NULL`.

vRefNum

The volume reference number, drive number, or 0 for the default volume.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
All files on the volume (except those opened by the Operating System) must be closed before you call `UnmountVol`, which does not eject the volume.

Most applications do not need to use this function, because the user typically ejects (and possibly also unmounts) a volume in the Finder.

Special Considerations

Don't unmount the startup volume. Doing so will cause a system crash.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Files.h`

Deprecated in Mac OS X v10.5

FlushVol

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `FSFlushVolume` (page 64) instead.)

```
OSErr FlushVol (
   ConstStr63Param volName,
   FSVolumeRefNum vRefNum
);
```

Parameters

volName

The name of the mounted volume to flush.

vRefNum

The volume reference number, drive number, or 0 for the default volume.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
For the specified volume, the `FlushVol` function writes the contents of the associated volume buffer and descriptive information about the volume. Information which has changed since the last time `FlushVol` was called is written to the volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
JPEG File Interchange Format
qtdataexchange
qtmoviefromprocs
qtmoviefromprocs.win
qtshortcut.win

Declared In
`Files.h`

FSpMakeFSRef

Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr FSpMakeFSRef (
   const FSSpec *source,
   FSRef *newRef
);
```

Parameters

source

A pointer to the `FSSpec` for the file or directory. This parameter must point to a valid `FSSpec` for an existing file or directory; if it does not, the call will return `fnfErr`. See `FSSpec` (page 226) for a description of the `FSSpec` data type.

newRef

On input, a pointer to an `FSRef` structure. On return, a pointer to the `FSRef` for the file or directory specified in the `FSSpec` pointed to in the `source` parameter.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
To obtain an `FSSpec` from an `FSRef`, use the `FSGetCatalogInfo` (page 67) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
ColorSyncDevices

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

DropDraw
FSCopyObject
ImageApp
MoreAppleEvents

Declared In
`Files.h`

PBCloseAsync

Closes an open file. (Deprecated in Mac OS X v10.5. Use `PBCloseForkAsync` (page 117) instead.)

```
OSErr PBCloseAsync (
    ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number to the file to close.

The `PBCloseAsync` function writes the contents of the access path buffer specified by the `ioRefNum` field to the volume and removes the access path.

Special Considerations

Some information stored on the volume won't be updated until `PBFlushVolAsync` is called.

Do not call `PBCloseAsync` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBCloseSync

Closes an open file. (Deprecated in Mac OS X v10.5. Use `PBCloseForkSync` (page 118) instead.)

```
OSErr PBCloseSync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a basic File Manager parameter block.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant field of the parameter block is:

`ioRefNum`

On input, a file reference number to the file to close.

The `PBCloseSync` function writes the contents of the access path buffer specified by the `ioRefNum` field to the volume and removes the access path.

Special Considerations

Some information stored on the volume won't be updated until `PBFlushVolSync` is called.

Do not call `PBCloseSync` with a file reference number of a file that has already been closed. Attempting to close the same file twice may result in loss of data on a volume.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBCreateFileIDRefAsync

Establishes a file ID reference for a file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr PBCreateFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `FIDParam` (page 204) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

Given a volume reference number, filename, and parent directory ID, the `PBCreateFileIDRefAsync` function creates a structure to hold the name and parent directory ID of the specified file. The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function. `PBCreateFileIDRefAsync` returns the result code `fidExists` if a file ID reference already exists for the file.

`ioNamePtr`
On input, a pointer to the file's name.

`ioVRefNum`
On input, a volume reference number for the volume containing the file.

`ioSrcDirID`
On input, the file's parent directory ID.

`ioFileID`
On output, a file ID. If a file ID reference already exists for the file, `PBCreateFileIDRefAsync` supplies the file ID but returns the result code `fidExists`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBCreateFileIDRefSync

Establishes a file ID reference for a file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr PBCreateFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `FIDParam` (page 204) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).`PBCreateFileIDRefSync` returns the result code `fidExists` if a file ID reference already exists for the file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

Given a volume reference number, filename, and parent directory ID, the `PBCreateFileIDRefSync` function creates a structure to hold the name and parent directory ID of the specified file. The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the file's name.

`ioVRefNum`
On input, a volume reference number for the volume containing the file.

`ioSrcDirID`
On input, the file's parent directory ID.

`ioFileID`
On output, a file ID. If a file ID reference already exists for the file, `PBCreateFileIDRefSync` supplies the file ID but returns the result code `fidExists`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBDeleteFileIDRefAsync

Deletes a file ID reference. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBDeleteFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to the `FIDParam` (page 204) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the file.

`ioFileID`
On input, the file ID reference to delete. After it has invalidated a file ID reference, the File Manager can no longer resolve that ID reference to a filename and parent directory ID.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBDeleteFileIDRefSync

Deletes a file ID reference. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBDeleteFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `FIDParam` (page 204) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the file.

`ioFileID`
On input, the file ID reference to delete. After it has invalidated a file ID reference, the File Manager can no longer resolve that ID reference to a filename and parent directory ID.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBFlushVolAsync

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `PBFlushVolumeAsync` (page 134) instead.)

```
OSErr PBFlushVolAsync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `VolumeParam` (page 259) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the volume to flush.

`ioVRefNum`

On input, a volume reference number, drive number, or 0 for the default volume.

`PBFlushVolAsync` flushes all open files on the volume, and then flushes all volume data structures. On the volume specified by `ioNamePtr` or `ioVRefNum`, the `PBFlushVolAsync` function writes descriptive information about the volume, the contents of the associated volume buffer, and all access path buffers for the volume (if they've changed since the last time `PBFlushVolAsync` was called).

The date and time of the last modification to the volume are set when the modification is made, not when the volume is flushed.

To ensure that all changes to a volume are flushed to the volume, use `PBFlushVolAsync`. You do not, however, need to flush a volume before unmounting it, ejecting it, or putting it offline; this is done automatically.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

If changes are made to a file that affect the file's end-of-file, the file's name, the file's Finder information, or the file's location on the volume, then you must use `PBFlushVolAsync`, or one of the other two volume flush functions in this section, to ensure that these changes are written to disk.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBFlushVolSync

Writes the contents of the volume buffer and updates information about the volume. (Deprecated in Mac OS X v10.5. Use `PBFlushVolumeSync` (page 134) instead.)

```
OSErr PBFlushVolSync (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `VolumeParam` (page 259) variant of the basic File Manager parameter block. See `ParamBlockRec` (page 252) for a description of the `ParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to the name of the volume to flush.

`ioVRefNum`
On input, the volume reference number, drive number, or 0 for the default volume.

`PBFlushVolSync` flushes all open files on the volume, and then flushes all volume data structures. On the volume specified by `ioNamePtr` or `ioVRefNum`, the `PBFlushVolSync` function writes descriptive information about the volume, the contents of the associated volume buffer, and all access path buffers for the volume (if they've changed since the last time `PBFlushVolSync` was called).

The date and time of the last modification to the volume are set when the modification is made, not when the volume is flushed.

To ensure that all changes to a volume are flushed to the volume, use `PBFlushVolSync`. You do not, however, need to flush a volume before unmounting it, ejecting it, or putting it offline; this is done automatically.

If changes are made to a file that affect the file's end-of-file, the file's name, the file's Finder information, or the file's location on the volume, then you must use `PBFlushVolSync`, or one of the other two volume flush functions in this section, to ensure that these changes are written to disk.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBGetVolMountInfo

Retrieves a record containing all the information needed to mount a volume, except for passwords. (Deprecated in Mac OS X v10.5. Use `FSVolumeMount` (page 106) instead.)

```
OSErr PBGetVolMountInfo (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to a pathname

`ioVRefNum`

On input, a volume specification. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`

On input, a pointer to a buffer to hold the mounting information. The length of the buffer is specified by the value pointed to by the `ioBuffer` field in a previous call to `PBGetVolMountInfoSize` (page 512). On output, the mounting information for the specified volume. You can later pass this structure to the `PBVolumeMount` (page 536) function to mount the volume. The mounting information for an AppleShare volume is stored as an AFP mounting record. The `PBGetVolMountInfo` function does not return the user password or volume password in the `AFPVolMountInfo` structure. Your application should solicit these passwords from the user and fill in the structure before attempting to mount the remote volume.

This function allows your application to record the mounting information for a volume and then to mount the volume later. This programmatic mounting function stores the mounting information in a structure called the `AFPVolMountInfo` (page 183) structure.

Special Considerations

This function executes synchronously. You should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

PBGetVolMountInfoSize

Determines how much space to allocate for a volume mounting information structure. (Deprecated in Mac OS X v10.5. Use `FSVolumeMount` (page 106) instead.)

```
OSErr PBGetVolMountInfoSize (
   ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to the `IOParam` (page 248) variant of the basic File Manager parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname

`ioVRefNum`
On input, a volume specification. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`
On input, a pointer to storage for the size information, which is of type `Integer` (2 bytes). If `PBGetVolMountInfoSize` returns `noErr`, that integer contains the size of the volume mounting information structure on output.

You should call this function before you call `PBGetVolMountInfo` (page 511) , to obtain the size of the volume mounting information for which you must allocate storage. Then call `PBGetVolMountInfo` to retrieve the actual volume mounting information.

Special Considerations

This function executes synchronously. You should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
FSMegaInfo

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

PBHCopyFileAsync

Duplicates a file and optionally renames it. (Deprecated in Mac OS X v10.5. Use `PBFSCopyFileAsync` (page 135) instead.)

```
OSErr PBHCopyFileAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to a `CopyParam` (page 191) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`

On output, the result code of the function.

`ioNamePtr`

On input, a pointer to the name of the source file.

`ioVRefNum`

On input, the volume reference number or drive number for the volume containing the source file. Pass 0 for the default volume.

`ioDstVRefNum`

On input, the reference number or drive number of the destination volume. Pass 0 for the default volume.

`ioNewName`

On input, a pointer to the partial pathname for the destination directory. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field.

`ioCopyName`

On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`

On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`

On input, the directory ID of the source directory.

This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Special Considerations

This is an optional call for AppleShare file servers. Your application should examine the information returned by the `PBHGetVolParmsAsync` (page 517) function to see if the volume supports `PBHCopyFileAsync`. If the `bHasCopyFile` bit is set in the `vMAttrib` field of the `GetVolParmsInfoBuffer` structure, then the volume supports `PBHCopyFileAsync`.

For AppleShare file servers, the source and destination pathnames must indicate the same file server; however, the parameter block may specify different source and destination volumes on that file server. A useful way to tell if two file server volumes are on the same file server is to call the `PBHGetVolParmsAsync` (page 517) function for each volume and compare the server addresses returned. The server opens source files with read/deny write enabled and destination files with write/deny read and write enabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHCopyFileSync

Duplicates a file and optionally renames it. (Deprecated in Mac OS X v10.5. Use `PBFSCopyFileSync` (page 135) instead.)

```
OSErr PBHCopyFileSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to a `CopyParam` (page 191) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

`ioNamePtr`

On input, a pointer to the name of the source file.

`ioVRefNum`

On input, the volume reference number or drive number for the volume containing the source file. Pass 0 for the default volume.

`ioDstVRefNum`

On input, the reference number or drive number of the destination volume. Pass 0 for the default volume.

`ioNewName`

On input, a pointer to the partial pathname for the destination directory. If `ioNewName` is `NULL`, the destination directory is the directory having the ID specified in the `ioNewDirID` field.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCopyName`
: On input, a pointer to the file's new name. The string pointed to by this field must be a filename, not a partial pathname. If you do not wish to rename the file, pass `NULL` in this field.

`ioNewDirID`
: On input, if the `ioNewName` field is `NULL`, the directory ID of the destination directory. If `ioNewName` is not `NULL`, the parent directory ID of the destination directory.

`ioDirID`
: On input, the directory ID of the source directory.

This function is especially useful when you want to copy or move files located on a remote volume, because it allows you to forgo transmitting large amounts of data across a network. This function is used internally by the Finder; most applications do not need to use it.

Special Considerations

This is an optional call for AppleShare file servers. Your application should examine the information returned by the `PBHGetVolParmsSync` (page 518) function to see if the volume supports `PBHCopyFileSync`. If the `bHasCopyFile` bit is set in the `vMAttrib` field of the `GetVolParmsInfoBuffer` structure, then the volume supports `PBHCopyFileSync`.

For AppleShare file servers, the source and destination pathnames must indicate the same file server; however, the parameter block may specify different source and destination volumes on that file server. A useful way to tell if two file server volumes are on the same file server is to call the `PBHGetVolParmsSync` (page 518) function for each volume and compare the server addresses returned. The server opens source files with read/deny write enabled and destination files with write/deny read and write enabled.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

PBHGetDirAccessAsync

Returns the access control information for a directory or file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr PBHGetDirAccessAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `AccessParam` (page 180) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname for the directory or file.

`ioVRefNum`
: On input, a volume specification for the volume containing the directory or file. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
: On output, the user ID for the owner of the directory or file.

`ioACGroupID`
: On output, the primary group ID of the directory or file.

`ioACAccess`
: On output, the access rights for the directory or file. See "File and Folder Access Privilege Constants" (page 297) for more information on these access rights.

`ioDirID`
: On input, the directory ID.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetDirAccessSync

Returns the access control information for a directory or file. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr PBHGetDirAccessSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `AccessParam` (page 180) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
On input, a pointer to a pathname for the directory or file.

`ioVRefNum`
On input, a volume specification for the volume containing the directory or file. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
On output, the user ID for the owner of the directory or file.

`ioACGroupID`
On output, the primary group ID of the directory or file.

`ioACAccess`
On output, the access rights for the directory or file. See "File and Folder Access Privilege Constants" (page 297) for more information on these access rights.

`ioDirID`
On input, the directory ID.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
FSCopyObject

Declared In
`Files.h`

PBHGetVolParmsAsync

Returns information about the characteristics of a volume. (Deprecated in Mac OS X v10.5. Use `FSGetVolumeParms` (page 76) instead.)

```
OSErr PBHGetVolParmsAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
On output, the result code of the function.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
: On input, a pointer to the volume's name. You can use a either a name or a volume specification to specify the volume. If you use a volume specification to specify the volume, you should set the `ioNamePtr` field to `NULL`.

`ioVRefNum`
: On input, a volume specification. You can use a either a name or a volume specification to specify the volume. A volume specification can be a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`
: On input, a pointer to a `GetVolParmsInfoBuffer` (page 233) record; you must allocate this memory to hold the returned attributes. On return, the `PBHGetVolParmsAsync` function places the attributes information in the buffer. Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer` structure. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear, then the volume does not implement the HFS Plus APIs and they are being emulated for that volume by the File Manager.

`ioReqCount`
: On input, the size, in bytes, of the buffer area pointed to in the `ioBuffer` field.

`ioActCount`
: On output, the size of the data actually returned.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHGetVolParmsSync

Returns information about the characteristics of a volume. (Deprecated in Mac OS X v10.5. Use `FSGetVolumeParms` (page 76) instead.)

```
OSErr PBHGetVolParmsSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `HIOParam` (page 241) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the volume's name. You can use a either a name or a volume specification to specify the volume. If you use a volume specification to specify the volume, you should set the `ioNamePtr` field to `NULL`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
: On input, a volume specification. You can use a either a name or a volume specification to specify the volume. A volume specification can be a volume reference number, drive number, or 0 for the default volume.

`ioBuffer`
: On input, a pointer to a `GetVolParmsInfoBuffer` (page 233) record; you must allocate this memory to hold the returned attributes. On return, the `PBHGetVolParmsSync` function places the attributes information in the buffer. Volumes that implement the HFS Plus APIs must use version 3 (or newer) of the `GetVolParmsInfoBuffer` structure. If the version of the `GetVolParmsInfoBuffer` is 2 or less, or the `bSupportsHFSPlusAPIs` bit is clear, then the volume does not implement the HFS Plus APIs and they are being emulated for that volume by the File Manager.

`ioReqCount`
: On input, the size, in bytes, of the buffer area pointed to in the `ioBuffer` field.

`ioActCount`
: On output, the size of the data actually returned.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
FSCopyObject
FSCreateFileAndOpenForkUnicode
FSMegaInfo
VolumeToBSDNode

Declared In
`Files.h`

PBHMapIDAsync

Determines the name of a user or group given the user or group ID. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapIDAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `ObjParam` (page 251) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname.

`ioVRefNum`
: On input, a volume specification.

`ioObjType`
: On input, the mapping function code its value is 1 if you're mapping a user ID to a user name or 2 if you're mapping a group ID to a group name. See "Mapping Code Constants" (page 312) for more information about the values you can use in this field.

`ioObjNamePtr`
: On output, a pointer to the user or group name; the maximum size of the name is 31 characters (preceded by a length byte).

`ioObjID`
: On input, the user or group ID to be mapped. AppleShare uses the value 0 to signify Any User.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might not need to specify a value in the `ioObjType` field.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHMapIDSync

Determines the name of a user or group given the user or group ID. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapIDSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `ObjParam` (page 251) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification.

`ioObjType`
On input, the mapping function code its value is 1 if you're mapping a user ID to a user name or 2 if you're mapping a group ID to a group name. See "Mapping Code Constants" (page 312) for more information about the values you can use in this field.

`ioObjNamePtr`
On output, a pointer to the user or group name; the maximum size of the name is 31 characters (preceded by a length byte).

`ioObjID`
On input, the user or group ID to be mapped. AppleShare uses the value 0 to signify Any User.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes
Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might not need to specify a value in the `ioObjType` field.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHMapNameAsync

Determines the user ID or group ID from a user or group name. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapNameAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to the `ObjParam` (page 251) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to a pathname.

`ioVRefNum`
: On input, a volume specification.

`ioObjType`
: On input, the mapping function code its value is 3 if you're mapping a user name to a user ID or 4 if you're mapping a group name to a group ID. See "Mapping Code Constants" (page 312) for more information on the values you can use in this field.

`ioObjNamePtr`
: On input, a pointer to the user or group name. The maximum size of the name is 31 characters. If `NULL` is passed, the ID returned is always 0.

`ioObjID`
: On output, the mapped user or group ID.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes

Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might need to set the `ioObjType` field to determine which database (user or group) to search first. If both a user and a group have the same name, this field determines which kind of ID you receive.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Files.h`

PBHMapNameSync

Determines the user ID or group ID from a user or group name. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBHMapNameSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to the `ObjParam` (page 251) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification.

`ioObjType`
On input, the mapping function code its value is 3 if you're mapping a user name to a user ID or 4 if you're mapping a group name to a group ID. See "Mapping Code Constants" (page 312) for more information on the values you can use in this field.

`ioObjNamePtr`
On input, a pointer to the user or group name. The maximum size of the name is 31 characters. If `NULL` is passed, the ID returned is always 0.

`ioObjID`
On output, the mapped user or group ID.

Special Considerations

See the BSD functions `getpwnam` and `getpwuid`, which correspond to this function on a conceptual level.

Version Notes
Because user and group IDs are interchangeable under AFP 2.1 and later volumes, you might need to set the `ioObjType` field to determine which database (user or group) to search first. If both a user and a group have the same name, this field determines which kind of ID you receive.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDenyAsync

Opens a file's data fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkAsync` (page 154) with deny modes in the permissions field.)

```
OSErr PBHOpenDenyAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic HFS parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

`ioNamePtr`
: On input, a pointer to a pathname for the file.

`ioVRefNum`
: On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
: On output, the file reference number for the file.

`ioDenyModes`
: On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 295) for a description of the types of access that you can request.

`ioDirID`
: On input, the parent directory ID of the file.

You should use the `PBHOpenDenyAsync` and `PBHOpenRFDenyAsync` (page 526) functions (or their synchronous counterparts, `PBHOpenDenySync` (page 524) and `PBHOpenRFDenySync` (page 527)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenDenyAsync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 295) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's resource fork with access deny permissions, use the `PBHOpenRFDenySync` (page 527) or `PBHOpenRFDenyAsync` (page 526) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenDenySync

Opens a file's data fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkSync` (page 155) with deny modes in the permissions field.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBHOpenDenySync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the AccessParam (page 180) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). The function returns the result code opWrErr if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in ioRefNum. You should not use this reference number unless your application originally opened the file.

Discussion

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to a pathname for the file.

ioVRefNum

On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

ioRefNum

On output, the file reference number for the file.

ioDenyModes

On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 295) for a description of the types of access that you can request.

ioDirID

On input, the parent directory ID of the file.

You should use the PBHOpenDenySync and PBHOpenRFDenySync (page 527) functions (or their asynchronous counterparts, PBHOpenDenyAsync (page 523) and PBHOpenRFDenyAsync (page 526)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. PBHOpenDenySync is not retried in any way. If the file cannot be opened because of a deny conflict, the error afpDenyConflict is returned and the ioRefNum field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the bHasOpenDeny bit is set in the vMAttrib field returned by the PBHGetVolParmsSync (page 518) or PBHGetVolParmsAsync (page 517) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 295) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's resource fork with access deny permissions, use the PBHOpenRFDenySync (page 527) or PBHOpenRFDenyAsync (page 526) function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Declared In
Files.h

PBHOpenRFDenyAsync

Opens a file's resource fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use PBOpenForkAsync (page 154) with deny modes in the permissions field.)

```
OSErr PBHOpenRFDenyAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the AccessParam (page 180) variant of the basic HFS parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329).

Discussion

The relevant fields of the parameter block are:

ioCompletion
On input, a pointer to a completion routine. For more information on completion routines, see IOCompletionProcPtr (page 179).

ioResult
On output, the result code of the function. The function returns the result code opWrErr if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in ioRefNum. You should not use this reference number unless your application originally opened the file.

ioNamePtr
On input, a pointer to a pathname for the file.

ioVRefNum
On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

ioRefNum
On output, the file reference number for the file.

ioDenyModes
On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 295) for a description of the types of access that you can request.

ioDirID
On input, the parent directory ID of the file.

You should use the PBHOpenRFDenyAsync and PBHOpenDenyAsync (page 523) functions (or their synchronous counterparts, PBHOpenRFDenySync (page 527) and PBHOpenDenySync (page 524)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. PBHOpenRFDenyAsync is not retried in any way. If the file cannot be opened because of a deny conflict, the error afpDenyConflict is returned and the ioRefNum field is set to 0.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 295) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's data fork with access deny permissions, use the `PBHOpenDenySync` (page 524) or `PBHOpenDenyAsync` (page 523) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHOpenRFDenySync

Opens a file's resource fork using the access deny modes. (Deprecated in Mac OS X v10.5. Use `PBOpenForkSync` (page 155) with deny modes in the permissions field.)

```
OSErr PBHOpenRFDenySync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to the `AccessParam` (page 180) variant of the basic HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329). The function returns the result code `opWrErr` if you've requested write permission and you have already opened the file for writing in that case, the existing file reference number is returned in `ioRefNum`. You should not use this reference number unless your application originally opened the file.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion function.

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname for the file.

`ioVRefNum`
On input, a volume reference number or drive number for the volume containing the file. Pass 0 to indicate the default volume.

`ioRefNum`
On output, the file reference number for the file.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioDenyModes`

On input, the type of access you are requesting to the fork. See "File Access Permission Constants" (page 295) for a description of the types of access that you can request.

`ioDirID`

On input, the parent directory ID of the file.

You should use the `PBHOpenRFDenySync` and `PBHOpenDenySync` (page 524) functions (or their asynchronous counterparts, `PBHOpenRFDenyAsync` (page 526) and `PBHOpenDenyAsync` (page 523)) if you want to ensure that you get the access permissions and deny-mode permissions that you request. `PBHOpenRFDenySync` is not retried in any way. If the file cannot be opened because of a deny conflict, the error `afpDenyConflict` is returned and the `ioRefNum` field is set to 0.

You can check that the volume supports AFP deny-mode permissions by checking that the `bHasOpenDeny` bit is set in the `vMAttrib` field returned by the `PBHGetVolParmsSync` (page 518) or `PBHGetVolParmsAsync` (page 517) function. If you don't want to special case volumes that support AFP deny mode permissions, you can use the File Manager permissions. See "File Access Permission Constants" (page 295) for a description of how File Manager permissions are translated to AFP deny-mode permissions.

To open a file's data fork with access deny permissions, use the `PBHOpenDenySync` (page 524) or `PBHOpenDenyAsync` (page 523) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetDirAccessAsync

Changes the access control information for a directory. (Deprecated in Mac OS X v10.5. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr PBHSetDirAccessAsync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`

A pointer to an `AccessParam` (page 180) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`

On input, a pointer to a completion function. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioResult`
On output, the result code of the function.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the directory. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
On input, the owner ID.

`ioACGroupID`
On input, the group ID.

`ioACAccess`
On input, the directory's access rights. You cannot set the owner or user rights bits of the `ioACAccess` field directly; if you try to do this, `PBHSetDirAccessAsync` returns the result code `paramErr`. Only the blank access privileges can be set for a directory using this function. See "File and Folder Access Privilege Constants" (page 297) for more information on directory access privileges.

`ioDirID`
On input, the directory ID.

To change the owner or group, you should set the `ioACOwnerID` or `ioACGroupID` field to the appropriate ID. You must be the owner of the directory to change the owner or group ID. A guest on a server can manipulate the privileges of any directory owned by the guest.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBHSetDirAccessSync

Changes the access control information for a directory. (Deprecated in Mac OS X v10.5. Use `FSSetCatalogInfo` (page 101) instead.)

```
OSErr PBHSetDirAccessSync (
   HParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to an `AccessParam` (page 180) variant of an HFS parameter block. See `HParamBlockRec` (page 243) for a description of the `HParamBlockRec` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioNamePtr`
On input, a pointer to a pathname.

`ioVRefNum`
On input, a volume specification for the volume containing the directory. This field can contain a volume reference number, drive number, or 0 for the default volume.

`ioACOwnerID`
On input, the owner ID.

`ioACGroupID`
On input, the group ID.

`ioACAccess`
On input, the directory's access rights. You cannot set the owner or user rights bits of the `ioACAccess` field directly; if you try to do this, `PBHSetDirAccessSync` returns the result code `paramErr`. Only the blank access privileges can be set for a directory using this function. See "File and Folder Access Privilege Constants" (page 297) for more information on directory access privileges.

`ioDirID`
On input, the directory ID.

To change the owner or group, you should set the `ioACOwnerID` or `ioACGroupID` field to the appropriate ID. You must be the owner of the directory to change the owner or group ID. A guest on a server can manipulate the privileges of any directory owned by the guest.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBMakeFSRefAsync

Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated in Mac OS X v10.5. Use `PBMakeFSRefUnicodeAsync` (page 151) instead.)

```
void PBMakeFSRefAsync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Discussion
For the parameter block based calls, the fields of the source `FSSpec` are passed as separate parameters (in the `ioNamePtr`, `ioVRefNum`, and `ioDirID` fields). This allows the call to be dispatched to external file systems the same way as other `FSp` calls are.

The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function.

`ioNamePtr`
: On input, a pointer to the name of the file or directory for which you wish to create an `FSRef`.

`ioVRefNum`
: On input, a volume specification for the volume containing the file or directory. This can be a volume reference number, a drive number, or 0 for the default volume.

`ioDirID`
: On input, the directory ID of the file or directory's parent directory.

`newRef`
: On input, a pointer to an `FSRef` structure. On output, this `FSRef` refers to the specified file or directory.

To obtain an `FSSpec` from an `FSRef`, use the `PBGetCatalogInfoAsync` (page 135) call.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBMakeFSRefSync

Creates an `FSRef` for a file or directory, given an `FSSpec`. (Deprecated in Mac OS X v10.5. Use `PBMakeFSRefUnicodeSync` (page 151) instead.)

```
OSErr PBMakeFSRefSync (
   FSRefParam *paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a file system reference parameter block. See `FSRefParam` (page 223) for a description of the `FSRefParam` data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
For the parameter block based calls, the fields of the source `FSSpec` are passed as separate parameters (in the `ioNamePtr`, `ioVRefNum`, and `ioDirID` fields). This allows the call to be dispatched to external file systems the same way as other `FSp` calls are.

The relevant fields of the parameter block are:

`ioNamePtr`
: On input, a pointer to the name of the file or directory for which you wish to create an `FSRef`.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioVRefNum`
On input, a volume specification for the volume containing the file or directory. This can be a volume reference number, a drive number, or 0 for the default volume.

`ioDirID`
On input, the directory ID of the file or directory's parent directory.

`newRef`
On input, a pointer to an `FSRef` structure. On output, this `FSRef` refers to the specified file or directory.

To obtain an `FSSpec` from an `FSRef`, use the `PBGetCatalogInfoSync` (page 139) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
AudioCDSample
FSCopyObject

Declared In
`Files.h`

PBReadAsync

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.5. Use `PBReadForkAsync` (page 158) instead.)

```
OSErr PBReadAsync (
    ParmBlkPtr paramBlock
);
```

Parameters
`paramBlock`
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for an open file to be read.

`ioBuffer`
On input, a pointer to a data buffer into which the bytes are read.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioReqCount`
: On input, the number of bytes requested. The value that you pass in this field should be greater than zero.

`ioActCount`
: On output, the number of bytes actually read.

`ioPosMode`
: On input, the positioning mode.

`ioPosOffset`
: On input, the positioning offset. On output, the new position of the mark.

This function attempts to read `ioReqCount` bytes from the open file whose access path is specified in the `ioRefNum` field and transfer them to the data buffer pointed to by the `ioBuffer` field. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If your application tries to read past the logical end-of-file, `PBReadAsync` reads the data, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `PBReadAsync` moves the file mark to the byte following the last byte read and returns `noErr`.

You can specify that `PBReadAsync` read the file data 1 byte at a time until the requested number of bytes have been read or until the end-of-file is reached. To do so, set bit 7 of the `ioPosMode` field. Similarly, you can specify that `PBReadAsync` should stop reading data when it reaches an application-defined newline character. To do so, place the ASCII code of that character into the high-order byte of the `ioPosMode` field; you must also set bit 7 of that field to enable newline mode.

When reading data in newline mode, `PBReadAsync` returns the newline character as part of the data read and sets `ioActCount` to the actual number of bytes placed into the buffer (which includes the newline character).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBReadSync

Reads any number of bytes from an open file. (Deprecated in Mac OS X v10.5. Use `PBReadForkSync` (page 159) instead.)

```
OSErr PBReadSync (
    ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
: A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

`ioRefNum`
On input, a file reference number for an open file to be read.

`ioBuffer`
On input, a pointer to a data buffer into which the bytes are read.

`ioReqCount`
On input, the number of bytes requested. The value that you pass in this field should be greater than zero.

`ioActCount`
On output, the number of bytes actually read.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

This function attempts to read `ioReqCount` bytes from the open file whose access path is specified in the `ioRefNum` field and transfer them to the data buffer pointed to by the `ioBuffer` field. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If your application tries to read past the logical end-of-file, `PBReadSync` reads the data, moves the mark to the end-of-file, and returns `eofErr` as its function result. Otherwise, `PBReadSync` moves the file mark to the byte following the last byte read and returns `noErr`.

You can specify that `PBReadSync` read the file data 1 byte at a time until the requested number of bytes have been read or until the end-of-file is reached. To do so, set bit 7 of the `ioPosMode` field. Similarly, you can specify that `PBReadSync` should stop reading data when it reaches an application-defined newline character. To do so, place the ASCII code of that character into the high-order byte of the `ioPosMode` field; you must also set bit 7 of that field to enable newline mode.

When reading data in newline mode, `PBReadSync` returns the newline character as part of the data read and sets `ioActCount` to the actual number of bytes placed into the buffer (which includes the newline character).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBResolveFileIDRefAsync

Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

```
OSErr PBResolveFileIDRefAsync (
   HParmBlkPtr paramBlock
);
```

Parameters
paramBlock
A pointer to an `FIDParam` (page 204) variant of the HFS parameter block. See `HParamBlockRec` (page 243) for more information on the `HParamBlockRec` data type.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDS internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

`ioCompletion`
: On input, a pointer to a completion routine. For more information on completion routines, see `IOCompletionProcPtr` (page 179).

`ioResult`
: On output, the result code of the function. A return code of `fidNotFound` means that the specified file ID reference has become invalid, either because the file was deleted or because the file ID reference was destroyed by `PBDeleteFileIDRefSync` (page 508) or `PBDeleteFileIDRefAsync` (page 507).

`ioNamePtr`
: On input, a pointer to a pathname. If the name string is `NULL`, `PBResolveFileIDRefAsync` doe s not return the filename, but returns only the parent directory ID of the file in the `ioSrcDirID` field. If the name string is not `NULL` but is only a volume name, `PBResolveFileIDRefAsync` ignores the value in the `ioVRefNum` field and uses the volume name instead. On output, a pointer to the filename for the file with the given file ID.

`ioVRefNum`
: On input, a volume specification for the volume containing the file. This field can contain a volume reference number, a drive number, or 0 for the default volume.

`ioSrcDirID`
: On output, the file's parent directory ID.

`ioFileID`
: On input, a file ID for the file to retrieve information about.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBResolveFileIDRefSync

Retrieves the filename and parent directory ID of a file given its file ID. (Deprecated in Mac OS X v10.5. Use `FSGetCatalogInfo` (page 67) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBResolveFileIDRefSync (
   HParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to an FIDParam (page 204) variant of the HFS parameter block. See HParamBlockRec (page 243) for more information on the HParamBlockRec data type.

Return Value

A result code. See "File Manager Result Codes" (page 329). A return code of fidNotFound means that the specified file ID reference has become invalid, either because the file was deleted or because the file ID reference was destroyed by PBDeleteFileIDRefSync (page 508) or PBDeleteFileIDRefAsync (page 507).

Discussion

Most applications do not need to use this function. In general, you should track files using alias records, as described in the Alias Manager documentation. The Alias Manager uses file IDs internally as part of its search algorithms for finding the target of an alias record.

The relevant fields of the parameter block are:

ioNamePtr

On input, a pointer to a pathname. If the name string is NULL, PBResolveFileIDRefSync doe s not return the filename, but returns only the parent directory ID of the file in the ioSrcDirID field. If the name string is not NULL but is only a volume name, PBResolveFileIDRefSync ignores the value in the ioVRefNum field and uses the volume name instead. On output, a pointer to the filename of the file with the given file ID.

ioVRefNum

On input, a volume specification for the volume containing the file. This field can contain a volume reference number, drive number, or 0 for the default volume.

ioSrcDirID

On output, the file's parent directory ID.

ioFileID

On input, a file ID for the file to retrieve information about.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Files.h

PBVolumeMount

Mounts a volume. (Deprecated in Mac OS X v10.5. Use FSVolumeMount (page 106) instead.)

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

```
OSErr PBVolumeMount (
   ParmBlkPtr paramBlock
);
```

Parameters

paramBlock

A pointer to the IOParam (page 248) variant of the basic File Manager parameter block. See HParamBlockRec (page 243) for a description of the HParamBlockRec data type.

Return Value
A result code. See "File Manager Result Codes" (page 329).

Discussion
The relevant fields of the parameter block are:

ioVRefNum

On output, a volume reference number for the mounted volume.

ioBuffer

On input, a pointer to mounting information. You can use the volume mounting information returned by the PBGetVolMountInfo (page 511) function or you can use a volume mounting information structure filled in by your application. If you're mounting an AppleShare volume, place the volume's AFP mounting information structure in the buffer pointed to by the ioBuffer field.

This function allows your application to record the mounting information for a volume and then to mount the volume later.

The PBGetVolMountInfo function does not return the user and volume passwords they're returned blank. Typically, your application asks the user for any necessary passwords and fills in those fields just before calling PBVolumeMount. If you want to mount a volume with guest status, pass an empty string as the user password.

If you have enough information about the volume, you can fill in the mounting structure yourself and call PBVolumeMount, even if you did not save the mounting information while the volume was mounted. To mount an AFP volume, you must fill in the structure with at least the zone name, server name, user name, user password, and volume password. You can lay out the fields in any order within the data field, as long as you specify the correct offsets.

In general, it is easier to mount remote volumes by creating and then resolving alias records that describe those volumes. The Alias Manager displays the standard user interface for user authentication when resolving alias records for remote volumes. As a result, this function is primarily of interest for applications that need to mount remote volumes with no user interface or with some custom user interface.

Special Considerations

AFP volumes currently ignore the user authentication method passed in the uamType field of the volume mounting information structure whose address is passed in the ioBuffer field of the parameter block. The most secure available method is used by default, except when a user mounts the volume as Guest and uses the kNoUserAuthentication authentication method.

This function executes synchronously. You should not call it at interrupt time.

Version Notes
The File Sharing workstation software introduced in system software version 7.0 does not currently pass the volume password. The AppleShare 3.0 workstation software does, however, pass the volume password.

Availability
Available in Mac OS X v10.0 and later.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBWaitIOComplete

Keeps the system idle until either an interrupt occurs or the specified timeout value is reached. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSErr PBWaitIOComplete (
    ParmBlkPtr paramBlock,
    Duration timeout
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

timeout
The maximum length of time you want the system to be kept idle.

Return Value
A result code. If the timeout value is reached, returns `kMPTimeoutErr`.

Special Considerations

This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBWriteAsync

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.5. Use `PBWriteForkAsync` (page 170) instead.)

```
OSErr PBWriteAsync (
    ParmBlkPtr paramBlock
);
```

Parameters

paramBlock
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioCompletion`
On input, a pointer to a completion routine.

`ioResult`
On output, the result code of the function.

`ioRefNum`
On input, a file reference number for the open file to which to write.

`ioBuffer`
On input, a pointer to a data buffer containing the bytes to write.

`ioReqCount`
On input, the number of bytes requested.

`ioActCount`
On output, the number of bytes actually written.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

The `PBWriteAsync` function takes `ioReqCount` bytes from the buffer pointed to by `ioBuffer` and attempts to write them to the open file whose access path is specified by `ioRefNum`. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If the write operation completes successfully, `PBWriteAsync` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `PBWriteAsync` moves the logical end-of-file. If you try to write past the physical end-of-file, `PBWriteAsync` adds one or more clumps to the file and moves the physical end-of-file accordingly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

PBWriteSync

Writes any number of bytes to an open file. (Deprecated in Mac OS X v10.5. Use `PBWriteForkSync` (page 171) instead.)

```
OSErr PBWriteSync (
    ParmBlkPtr paramBlock
);
```

Parameters

`paramBlock`
A pointer to a basic File Manager parameter block.

Return Value
A result code. See "File Manager Result Codes" (page 329).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Discussion
The relevant fields of the parameter block are:

`ioRefNum`
On input, a file reference number for the open file to which to write.

`ioBuffer`
On input, a pointer to a data buffer containing the bytes to write.

`ioReqCount`
On input, the number of bytes requested.

`ioActCount`
On output, the number of bytes actually written.

`ioPosMode`
On input, the positioning mode.

`ioPosOffset`
On input, the positioning offset. On output, the new position of the mark.

The `PBWriteSync` function takes `ioReqCount` bytes from the buffer pointed to by `ioBuffer` and attempts to write them to the open file whose access path is specified by `ioRefNum`. The position of the mark is specified by `ioPosMode` and `ioPosOffset`. If the write operation completes successfully, `PBWriteSync` moves the file mark to the byte following the last byte written and returns `noErr`. If you try to write past the logical end-of-file, `PBWriteSync` moves the logical end-of-file. If you try to write past the physical end-of-file, `PBWriteSync` adds one or more clumps to the file and moves the physical end-of-file accordingly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Files.h`

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *File Manager Reference*.

Date	Notes
2009-11-17	Added several constants.
	Added constants introduced in Mac OS X v10.4 for "Extended Volume Attributes" (page 289).
2007-07-13	Made minor format changes and added new content.
	Added information about high-level operations for moving objects to the Trash.
2006-09-05	Added information about high-level file operations. Removed constants and data types not declared in Files.h.
2006-03-08	Added deprecation information.
	Added field descriptions to `HParamBlockRec` (page 243).
	Updated description of the file access permission constant `fsRdWrShPerm` (page 296).
	Updated descriptions of several fields in the `FSCatalogInfo` (page 212) structure.
2005-08-11	Updated to fix several documentation bugs.
2005-04-29	Added new information to discussion of FSGetVolumeInfo function.
2004-03-01	Moved functions that are no longer recommended to the section "Not Recommended" (page 36).
	Changed the discussion of `FSGetCatalogInfoBulk` (page 68).
	Updated the description of the *`whichInfo`* parameter to the functions `FSSetCatalogInfo` (page 101), `PBSetCatalogInfoSync` (page 163), and `PBSetCatalogInfoAsync` (page 162).
	Added constant descriptions to "Extended Volume Attributes" (page 289).
	Updated the discussion for the `PBLockRangeSync` (page 476) and `PBLockRangeAsync` (page 474) functions.
	Corrected the description of the *`ioVDrvInfo`* argument to `PBHGetVInfoSync` (page 450) and `PBHGetVInfoAsync` (page 447).

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.

Date	Notes
	Corrected the description of the *driveNumber* field of the FSVolumeInfo (page 228) structure.
	Updated discussion of several functions in the section "Getting and Setting Volume Information" (page 26).
2003-02-01	Updated formatting.

2009-11-17 | © 2001, 2009 Apple Inc. All Rights Reserved.