Apple Filing Protocol Reference

Networking > Mac OS X Server

2006-05-23

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

Contents

Figures and Tables

Apple Filing Protocol Reference 11

Apple Filing Protocol Reference

Companion guide Apple Filing Protocol Programming Guide

Overview

This document describes the Apple Filing Protocol (AFP) commands, data types and constants that can be used to communicate with an AFP file server. AFP allows users of multiple computers to share files easily and efficiently over a network.

Functions

FPAccess

Requests access to a file or directory on a volume for which ACLs are enabled.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
unsigned short Bitmap
16 bytes UUID
long ReqAccess
byte Pathtype
string Pathname
```

Parameters

CommandCode

`kFPAccess` (75).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Reserved.

UUID

Universally Unique Identifier (UUID) of the process sending this command.

ReqAccess

Requested access. For definitions, see "ACL Access Rights" (page 174).

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the file or directory for which access is being requested. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if access is allowed. See Table 2 (page 15) for other possible result codes.

Discussion

The request is sent to the server, which determines whether to grant access.

Support for this command, as well as `FPGetACL` (page 70) and `FPSetACL` (page 140) is required in order to support access control lists (ACLs). Support for UTF-8 and UUIDs is also required in order to support ACLs.

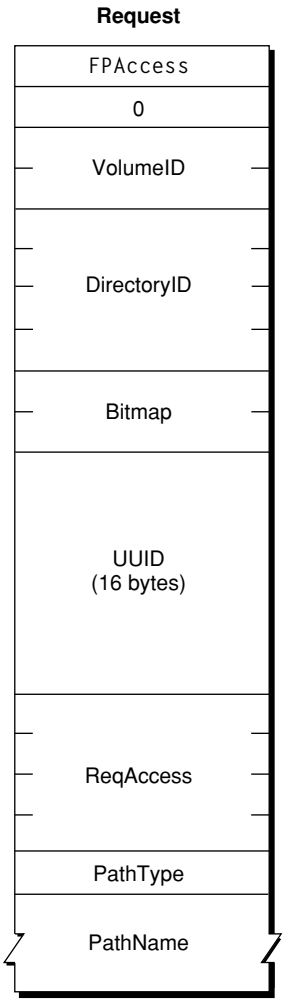
Table 2 (page 15) lists the result codes for the `FPAccess` command.

Table 1 Result codes for the `FPAccess` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to request access to the file or directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

Figure 1 shows the request block for the `FPAccess` command.

Figure 1 Request block for the `FPAccess` command



Version Notes
Introduced in AFP 3.2.

FPAddAPPL

Adds an APPL mapping to the Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long DirectoryID
long FileCreator
long ApplTag
byte PathType
string Pathname
```

Parameters

CommandCode

kFPAddAPPL (53).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

DirectoryID

Ancestor Directory ID.

FileCreator

File creator of the application corresponding to the APPL mapping being added.

ApplTag

User-defined tag stored with the APPL mapping.

PathType

Type of names in Pathname. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the application corresponding to the APPL mapping being added. Pathname is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

kFPNoErr if no error occurred. See Table 2 (page 15) for other possible result codes.

Reply block

None.

Discussion

This command adds the specified mapping to the volume's Desktop database, including the application's location, and file creator. If an APPL mapping for the same application (same filename, same directory, and same file creator) already exists, the mapping is replaced.

The user must have search or write privileges to all ancestors except the application's parent directory, as well as write access to the parent directory.

There may be more than one application in the Desktop database's list of APPL mappings for a given file creator. To distinguish among them, the ApplTag parameter is stored with each APPL mapping. The tag information may be used to decide among these multiple applications and is not interpreted by the Desktop database.

The user must have previously called FPOpenDT (page 120) for the corresponding volume. In addition, the application must be present in the specified directory before this command is sent.

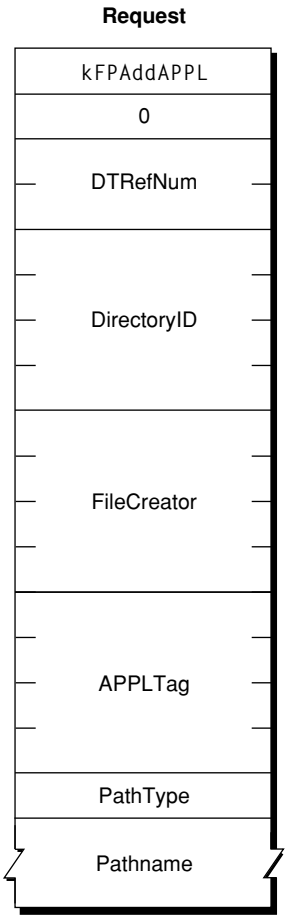
Table 2 (page 15) lists the result codes for the FPAddAPPL command.

Table 2 Result codes for the `FPAddAPPL` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to add an APPL mapping.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file.
`kFPObjectTypeErr`	Input parameters point to a directory.
`kFPParamErr`	Session reference or Desktop database reference number is unknown; pathname is invalid.

Figure 2 shows the request block for the `FPAddAPPL` command.

Figure 2 Request block for the `FPAddAPPL` command



FPAddComment

Adds a comment for a file or directory to a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long DirectoryID
byte PathType
string Pathname
string Comment
```

Parameters

CommandCode

`kFPAddComment` (56).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

DirectoryID

Ancestor Directory ID.

PathType

Type of name in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the file or directory with which the comment is to be associated. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Comment

Comment data to be associated with the specified file or directory.

Result

`kFPNoErr` if no error occurred. See Table 3 (page 17) for other possible result codes.

ReplyBlock

None.

Discussion

This command stores the comment data in the Desktop database and associates the comment with the specified file or directory. If the comment is longer than 199 bytes, the comment is truncated to 199 bytes without returning an error.

To add a comment to a directory that is not empty, the user needs search access to all ancestors including the directory's parent directory, as well as write access to the parent directory. To add a comment to an empty directory, the user needs search or write access to all ancestors except the directory's parent directory, as well as write access to the parent directory.

To add a comment to a file that is not empty, the user needs search access to all ancestors except the file's parent directory, as well as read and write access to the parent directory. To add a comment to an empty file, the user needs search or write access to all ancestors except the files's parent directory, as well as write access to the parent directory.

The user must have previously called `FPOpenDT` (page 120) for the corresponding volume. In addition, the specified file or directory must be present in the specified directory before this command is sent.

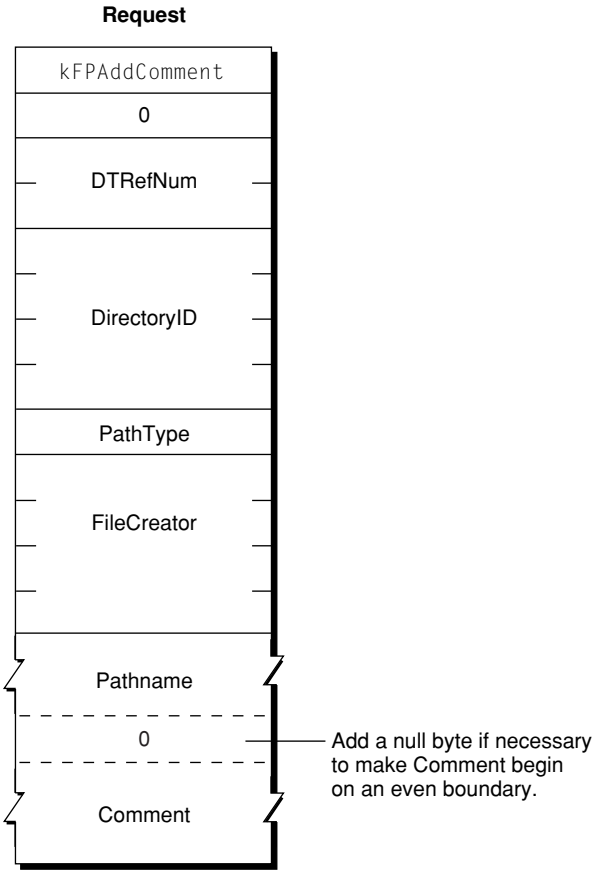
Table 3 (page 17) lists the result codes for the `FPAddComment` command.

Table 3 Result codes for the `FPAddComment` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to add a comment.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPMiscErr`	Non-AFP error occurred.

Figure 3 (page 17) shows the request block for the `FPAddComment` command.

Figure 3 Request block for the `FPAddComment` command



FPAddIcon

Adds an icon bitmap to a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long FileCreator
long FileType
byte IconType
byte Pad
long IconTag
short BitmapSize
```

Parameters

CommandCode

kFPAddIcon (192).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

FileCreator

File creator associated with the icon that is to be added.

FileType

File type associated with the icon that is to be added.

IconType

Type of icon that is to be added.

Pad

Pad byte.

IconTag

Tag information to be stored with the icon.

BitmapSize

Size of the bitmap for this icon.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or the Desktop database reference number is unknown or if the pathname is invalid, kFPIconTypeError if the new icon's size is different from the size of the existing icon, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

None.

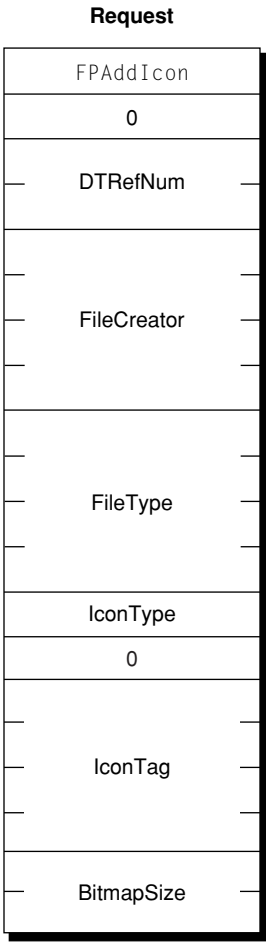
Discussion

This command adds the icon for the specified file creator and icon type to the Desktop database and associates the tag information with the icon. If an icon of the same file creator and icon type already exists in the database, the icon is replaced. However, if the new icon's size is different from the old icon, the server returns a kFPIconTypeError result code.

The user must have previously called FPOpenDT (page 120) for the corresponding volume.

Figure 4 (page 19) shows the request block for the FPAddIcon command.

Figure 4 Request block for the `FPAddIcon` command



FPByteRangeLock

Locks or unlocks a specified range of bytes within an open fork.

```
byte CommandCode
byte Flags
short OForkRefNum
long Offset
long Length
```

Parameters

CommandCode

`kFPByteRangeLock` (1).

Pad

Pad byte.

DTRefNum

Bit 0 is the LockUnlock bit, where 0 indicates lock and 1 indicates unlock. Bit 7 is the StartEndFlag bit, where 0 indicates that Offset is relative to the beginning of the fork and 1 indicates that Offset is relative to the end of the fork. The StartEndFlag bit is only used when locking a range.

OForkRefNum

Open fork reference number.

Offset

Offset to the first byte of the range to be locked or unlocked (can be negative if the StartEndFlag bit is set to 1).

Length

Number of bytes to be locked or unlocked (a signed, positive long integer; cannot be negative except for the special value $FFFFFFFF).

Result

kFPNoErr if no error occurred. See Table 4 (page 21) for possible result codes.

ReplyBlock

If the result code is kFPNoErr and the reply is for an attempt to lock a range, the server returns a reply block. The reply block consists of a long, called RangeStart, containing the number of the first byte of the range that was locked.

Discussion

This command locks and unlocks the specified range of bytes within an open fork for use by a user application. When locking a range, the server returns the number of the first locked byte.

Bytes are numbered from 0 to $7FFFFFFF. The latter value is the maximum size of the fork. The end of fork is one more than the number of the last byte in the fork.

If no user holds a lock on any part of the requested range, the server locks the range specified by this command. A user can hold multiple locks within the same open fork, up to a server-specific limit. Locks cannot overlap. A locked range can start or extend past the end of fork; this does not move the end of fork or prevent another user from writing to the fork past the locked range. Setting Offset to zero, the StartEndFlag bit to zero (start), and Length to $FFFFFFFF locks the entire fork to the maximum size of the fork. Setting Offset to a value other than zero, the StartEndFlag bit to zero, and Length to $FFFFFFFF locks a range beginning at Offset and extending to the maximum size of the fork.

Setting the StartEndFlag bit to 1 (end) allows a lock to be offset relative to the end of the fork. This enables a user to set a lock when the user does not know the exact end of the fork, as can happen when multiple writers are concurrently modifying the fork. The server returns the number of the first locked byte.

Lock conflicts are determined by the value of OForkRefNum. That is, if a fork is opened twice, the two open fork reference numbers are considered two different “users” regardless of whether they were opened for the same or different sessions.

All locks held by a user are unlocked when the user closes the fork. Unlocking a range makes it available to other users for reading and writing. The server returns a result code of kFPRangeNotLocked if a user tries to unlock a range that was locked by another user or that was not locked at all.

To unlock a range, the StartEndFlag bit must be set to zero (start), Length must match the size of the range that was locked, and Offset must match the number of the first byte in the locked range. If the range was locked with the StartEndFlag bit set to zero (start), use the same value of Offset to unlock the range that was used to lock the range. If the range was locked with the StartEndFlag bit set to 1 (end), set Offset to the value of RangeStart that was returned by the server. You cannot unlock part of range.

Mac OS X supports memory-mapped files, but byte range locks should not be used in conjunction with them.

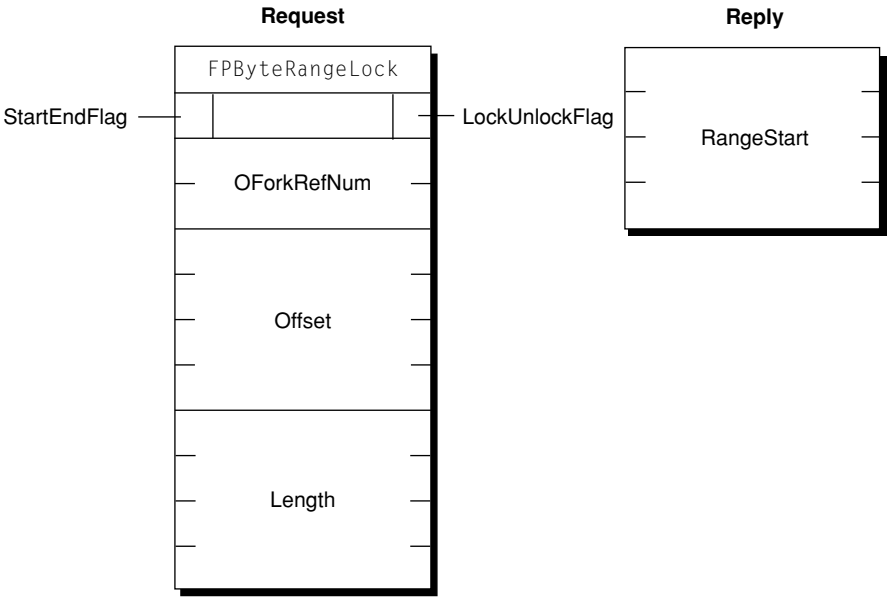
Table 4 (page 21) lists the result codes for the `FPByteRangeLock` command.

Table 4 Result codes for the `FPByteRangeLock` command

Result code	Explanation
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPNoMoreLocks`	Server's maximum lock count has been reached.
`kFPParamErr`	Session reference number or open fork reference number is unknown; a combination of the StartEndFlag bit and Offset specifies a range that starts before byte zero.
`kFPRangeNotLocked`	User tried to unlock a range that is locked by another user or that is not locked at all.
`kFPRangeOverlap`	User tried to lock some or all of a range that the user has already locked.

Figure 5 (page 21) shows the request and reply blocks for the `FPByteRangeLock` command.

Figure 5 Request and reply blocks for the `FPByteRangeLock` command



FPByteRangeLockExt

Locks or unlocks a specified range of bytes within an open fork.

```
byte CommandCode
byte Flags
short OForkRefNum
long long Offset
long long Length
```

Parameters

CommandCode

kFPByteRangeLockExt (59).

Pad

Pad byte.

Flags

Bit 0 is the LockUnlock bit, where 0 indicates lock and 1 indicates unlock. Bit 7 is the StartEndFlag bit, where 0 indicates that Offset is relative to the beginning of the fork and 1 indicates that Offset is relative to the end of the fork. The StartEndFlag bit is only used when locking a range.

OForkRefNum

Open fork reference number.

Offset

Offset to the first byte of the range to be locked or unlocked (can be negative if the StartEndFlag bit is set to 1).

Length

Number of bytes to be locked or unlocked (a signed, positive long integer; cannot be negative except for the special value $FFFFFFFFFFFFFFFF).

Result

kFPNoErr if no error occurred. See Table 5 (page 23) for possible result codes.

ReplyBlock

If the result code is kFPNoErr and the reply is for an attempt to lock a range, the server returns a reply block. The reply block consists of a long, called RangeStart, containing the number of the first byte of the range that was locked.

Discussion

This command locks and unlocks the specified range of bytes within an open fork for use by a user application. When locking a range, the server returns the number of the locked byte.

The FPByteRangeLockExt command differs from the FPByteRangeLock command in that the FPByteRangeLockExt command is prepared to handle large values that may be required for locking ranges for volumes larger than 4 GB in size.

Bytes are numbered starting from 0. The end of fork is one more than the number of the last byte in the fork.

If no user holds a lock on any part of the requested range, the server locks the range specified by this command. A user can hold multiple locks within the same open fork, up to a server-specific limit. Locks cannot overlap. A locked range can start or extend past the end of the fork; this does not move the end of the fork or prevent another user from writing to the fork past the locked range. Setting Offset to zero, the StartEndFlag bit to zero (start), and Length to $FFFFFFFFFFFFFFFF locks the entire for to the maximum size of the fork. Specifying an offset other than zero, the StartEndFlag bit to zero (start), and Length to $FFFFFFFFFFFFFFFF locks a range beginning at Offset and extending to the maximum size of the fork.

Setting the StartEndFlag bit to 1 (end) allows a lock to be offset relative to the end of the fork. This enables a user to set a lock when the user does not know the exact end of the fork, as can happen when multiple writers are concurrently modifying the fork. The server returns the number of the first locked byte.

Lock conflicts are determined by the value of `OForkRefNum`. That is, if a fork is opened twice, the two open fork reference numbers are considered two different "users" regardless of whether they were opened for the same or different sessions.

All locks held by a user are unlocked when the user closes the fork. Unlocking a range makes it available to other users for reading and writing. The server returns a result code of `kFPRangeNotLocked` if a user tries to unlock a range that was locked by another user or that was not locked at all.

To unlock a range, the `StartEndFlag` bit must be set to zero (start), `Length` must match the size of the range that was locked, and `Offset` must match the number of the first byte in the locked range. If the range was locked with `StartEndFlag` set to zero (start), use the same value of `Offset` to unlock the range that was used to lock the range. If the range was locked with the `StartEndFlag` bit set to 1 (end), set `Offset` to the value of `RangeStart` that was returned by the server. You cannot unlock part of range.

Mac OS X supports memory-mapped files, but byte range locks should not be used in conjunction with them.

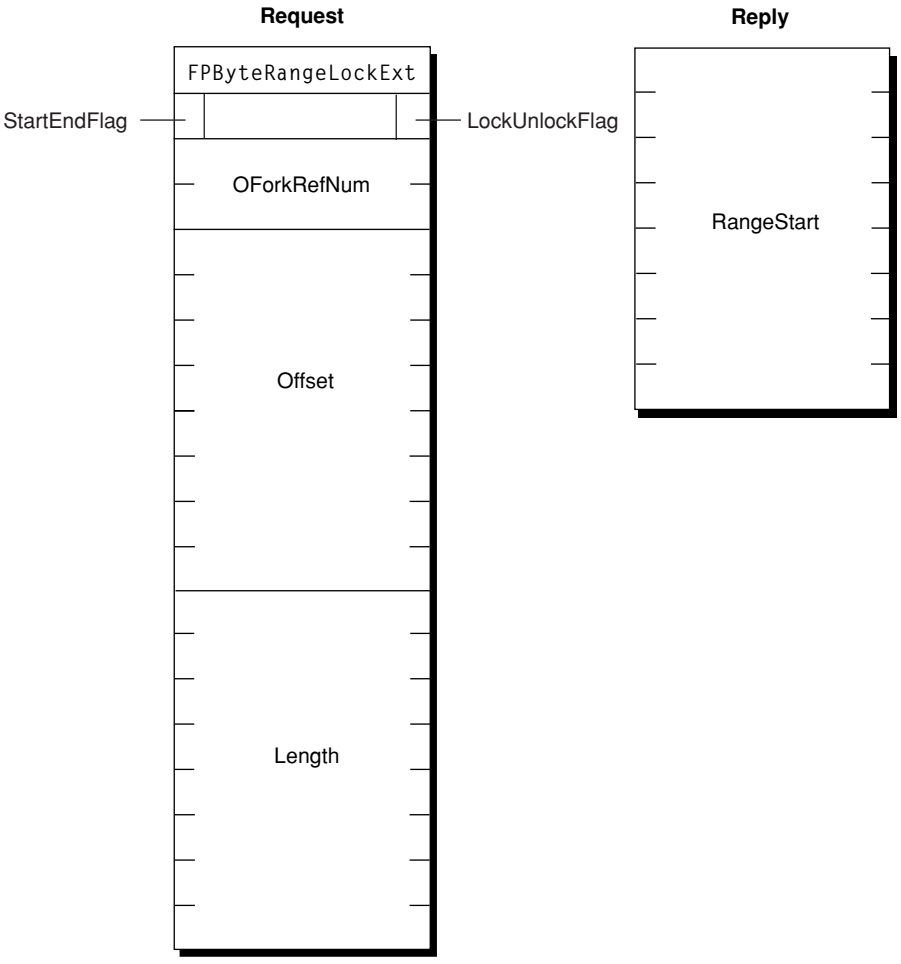
Table 5 (page 23) lists the result codes for the `FPByteRangeLockExt` command.

Table 5 Result codes for the `FPByteRangeLockExt` command

Result code	Explanation
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPNoMoreLocks`	Server's maximum lock count has been reached.
`kFPParamErr`	Open fork reference number is unknown; a combination of the StartEndFlag bit and Offset parameters specifies a range that starts before byte zero.
`kFPRangeOverlap`	User tried to lock some or all of a range that the user has already locked.
`kFPRangeNotLocked`	User tried to unlock a range that is locked by another user or that is not locked at all.

Figure 6 (page 24) shows the request and reply blocks for the `FPByteRangeLockExt` command.

Figure 6 Request and reply blocks for the `FPByteRangeLockExt` command



FPCatSearch

Searches a volume for files and directories that match specified criteria.

```
byte CommandCode
byte Pad
short VolumeID
long ReqMatches
long Reserved
16 bytes CatalogPosition
short FileRsltBitmap
short DirectoryRsltBitmap
long ReqBitmap
Specification1
Specification2
unsigned char Length
```

Parameters

CommandCode

`kFPCatSearch` (43).

Pad

Pad byte.

VolumeID

The ID of the volume to search.

Reserved

Reserved; must be zero.

ReqMatches

The maximum number of matches to return.

CatalogPosition

Current position in the catalog.

FileRsltBitmap

File bitmap describing the file parameters to get or null to get directory parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command with some restrictions described in the Discussion section. For bit definitions for this bitmap, see File Bitmap (page 164).

DirectoryRsltBitmap

Directory bitmap describing the directory parameters to get or null to get file parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command with some restrictions described in the Discussion section. For bit definitions for this bitmap, see Directory Bitmap (page 162).

ReqBitmap

Directory and file parameters that are to be searched. For directory parameters only, see Figure 7 (page 27). For file parameters only, see Figure 8 (page 27). For directory and file parameters, see Figure 9 (page 27).

Specification1

Search criteria lower bounds and values.

Specification2

Optional search criteria upper bounds and masks.

Length

Length of this request block.

Result

`kFPNoErr` if no error occurred. See Table 6 (page 28) for possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 7 (page 28) for the format of the reply block.

Discussion

This command searches a volume for files and directories that match the specified criteria and returns an array of records that describe the matches that were found. The criteria can include most parameters in the File bitmap, the Directory bitmap, or both bitmaps, that are defined for the FPGetFileDirParms (page 80) command. Parameters for the matching files and directories are returned. These parameters can also be any of those specified by the FPGEtFileDirParms command.

The first word of the CatalogPosition parameter specifies whether the parameter denotes an actual catalog position or a hint. If the first word is zero, the search starts at the beginning of the volume. If the first word is non-zero, CatalogPosition is a actual catalog position and the search starts with this entry.

The Specification1 and Specification2 parameters are used together to specify the search parameters. These parameters are packed in the same order as the bits in ReqBitmap. All variable-length parameters (such as those containing names) are put at the end of each specification record. An offset is stored in the specification parameters to indicate where the actual variable-length parameter is located. This offset is measured from the start of the specification parameters (not including the length and filler bytes). Results are packed in the same way.

The fields in Specification1 and Specification2 have different uses:

- In the name field, Specification1 holds the target string; Specification2 must always have a null name field.
- In all date and length fields, Specification1 holds the lowest value in the target range and Specification2 holds the highest value in the target range.
- In file attributes and Finder Info fields, Specification1 holds the target value and Specification2 holds the bitwise mask that specifies which bits in that field in Specification1 are relevant to the current search.

This command returns a result code of kFPEOFErr only when it has reached the end of the volume directory tree. For example, if the client requests ten matches, the server may return only four matches, without returning an error. The client should then send a request for six (ten minus four) more matches, using the same CatalogPosition value that was received in the previous reply. This process continues until the originally requested matches are received or a kFPEOFErr is returned. If this command returns a result code of kFPCatalogChanged, the client cannot continue the search. The client must restart the search by setting the first word of CatalogPosition to zero.

This command returns parameters for files, directories or both, depending on the value of the FileRsltBitmap and DirectoryRsltBitmap parameters. If FileRsltBitmap is null, this command assumes that you are not searching for files. Likewise, if DirectoryRsltBitmap is null, this command assumes that you are not searching for directories. If both parameters are non-zero, this command searches for files and directories. Note that if you are searching for both files and directories, certain restrictions apply with regard to the parameters that are searched. The rest of this section describes these restrictions.

The ReqBitmap parameter specifies the directory and file parameters to be searched. The low-order word of ReqBitmap is the same as low-order word of the File bitmap and the Directory bitmap used by the FPGetFileDirParms (page 80) command, with the exception of the Short Name parameter, which cannot be searched. The high bit of the high-order word of ReqBitmap indicates whether the search should match on full names or partial names (0 = full name, 1 = partial name). There is no equivalent to the fsSBNegate bit used by the Macintosh File Manager's PBCatSearch function.

Figure 7 (page 27) shows parameters this command can search when it is searching directories only.

Figure 7 Parameters `FPCatSearch` searches when searching directories only

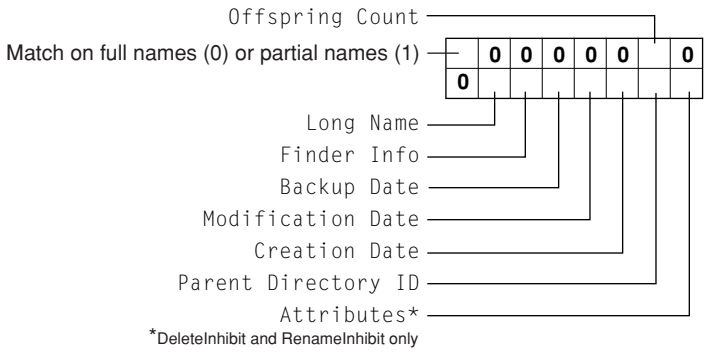


Figure 8 (page 27) shows parameters this command can search when it is searching files only.

Figure 8 Parameters `FPCatSearch` searches when searching files only

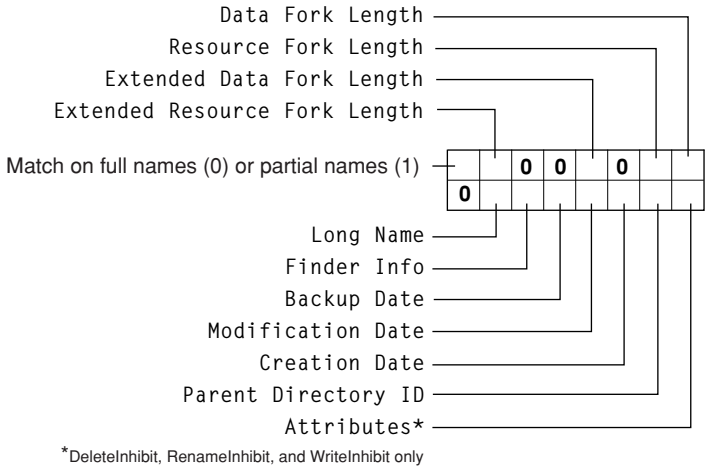
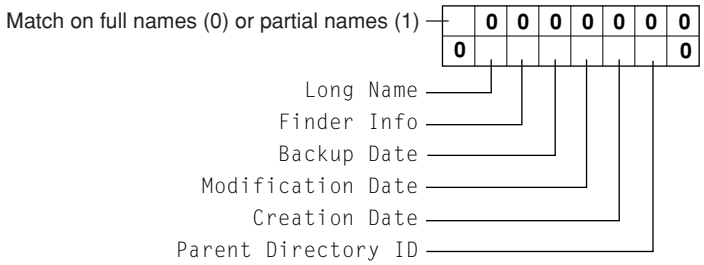


Figure 9 (page 27) shows parameters this command can search when it is searching both directories and files.

Figure 9 Parameters `FPCatSearch` searches when searching directories and files



Before sending this command, the user must call `FPOpenVol` (page 124) for the volume that is to be searched.

To return all files and directories that match the specified criteria, the user must have Read Only or Read & Write privileges for all directories. This command skips directories for which the user does not have Read Only or Read & Write privileges.

Table 6 lists the result codes for the `FPCatSearch` command.

Table 6 Result codes for the `FPCatSearch` command

Result code	Explanation
`kFPCallNotSupported`	Server does not support this command.
`kFPCatalogChanged`	Catalog has changed and CatalogPosition may be invalid. No matches were returned.
`kFPEOFErr`	No more matches.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is null or bad.

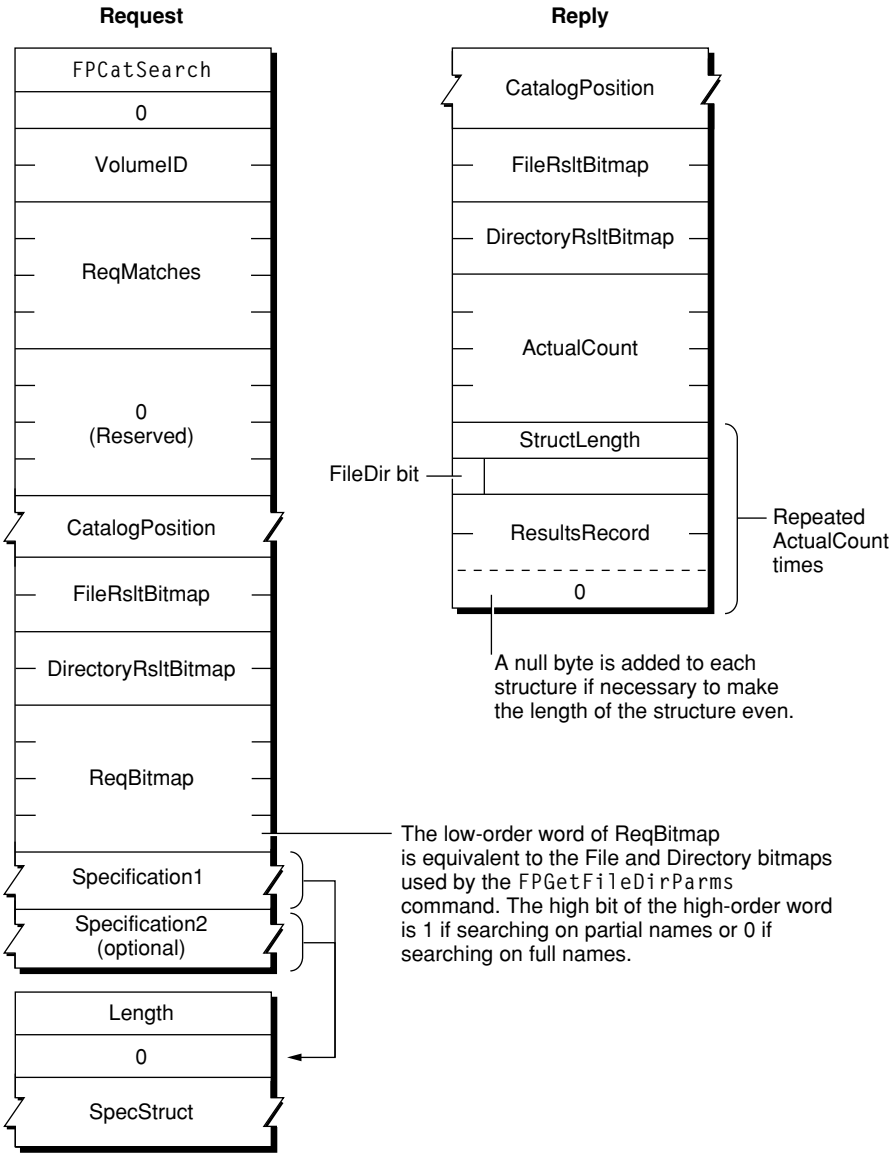
Table 7 describes the reply block for the `FPCatSearch` command.

Table 7 Reply block for the `FPCatSearch` command

Name and size	Data
`CatalogPosition` (16 bytes)	Current position in the catalog.
`FileRsltBitmap` (short)	Copy of the input bitmap.
`DirectoryRsltBitmap` (short)	Copy of the input bitmap.
`ActualCount` (byte)	Number of ResultsRecord structures that follow.
Zero or more `ResultsRecord` structures	Array of `ResultsRecord` structures describing the matches that were found and having the following structure: `StructLength` (byte) — Unsigned length of this structure including this byte and the byte for the `FileDir` bit. `FileDir` (bit 7 of a one-byte value) — Whether the record is for a file (0) or directory (1). `Results` — The matching Long Name, Parent Directory ID, or both with a trailing null byte if necessary to make the entire structure end on an even boundary.

Figure 10 shows the request and reply blocks for the `FPCatSearch` command.

Figure 10 Request and reply blocks for the `FPCatSearch` command



FPCatSearchExt

Searches a volume for files and directories that match specified criteria.

```
byte CommandCode
byte Pad
short VolumeID
long ReqMatches
long Reserved
16 bytes CatalogPosition
short FileRsltBitmap
short DirectoryRsltBitmap
long ReqBitmap
Specification1
Specification2
unsigned char Length
```

Parameters

CommandCode

`kFPCatSearchExt` (67).

Pad

Pad byte.

VolumeID

Volume ID.

ReqMatches

Maximum number of matches to return.

Reserved

Reserved; must be zero.

CatalogPosition

Current position in the catalog.

FileRsltBitmap

Bitmap describing the file parameters to get or null to get directory parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command with some restrictions described later in this section. For bit definitions for this bitmap, see File Bitmap (page 164).

DirectoryRsltBitmap

Bitmap describing the directory parameters to get or null to get file parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command with some restrictions described later in this section. For bit definitions for this bitmap, see Directory Bitmap (page 162).

ReqBitmap

Directory and file parameters that are to be searched. For directory parameters, see Figure 11 (page 32). For file parameters, see Figure 12 (page 32). For directory and file parameters, see Figure 13 (page 33).

Specification1

Search criteria lower bounds and values.

Specification2

Optional search criteria upper bounds and masks.

Length

Length of this request block.

Result

`kFPNoErr` if no error occurred. See Table 8 (page 33) for possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 9 (page 33) for the format of the reply block.

Discussion

This command searches a volume for files and directories that match the specified criteria and returns an array of records that describe the matches that were found.

This command differs from the FPCatSearch (page 24) command in that FPCatSearchExt is prepared to handle longer search results that can occur when searching volumes that are more than 4 GB in size.

The criteria can include most parameters in the File bitmap, the Directory bitmap, or both bitmaps, that are defined for the FPGetFileDirParms (page 80) command. Parameters for the matching files and directories are returned. These parameters can also be any of those specified by the FPGetFileDirParms command.

The first word of the CatalogPosition parameter specifies whether the parameter denotes an actual catalog position or a hint. If the first word is zero, the search starts at the beginning of the volume. If the first word is non-zero, CatalogPosition is an actual catalog position and the search starts with this entry.

The Specification1 and Specification2 parameters are used together to specify the search parameters. These parameters are packed in the same order as the bits in the ReqBitmap. All variable-length parameters (such as those containing names) are put at the end of each specification record. An offset is stored in the specification parameters to indicate where the actual variable-length parameter is located. This offset is measured from the start of the specification parameters (not including the length and filler bytes). Results are packed in the same way.

The fields in Specification1 and Specification2 have different uses:

- In the name field, Specification1 holds the target string; Specification2 must always have a null name field.
- In all date and length fields, Specification1 holds the lowest value in the target range and Specification2 holds the highest value in the target range.
- In Attributes and Finder Info fields, Specification1 holds the target value and Specification2 holds the bitwise mask that specifies which bits in that field in Specification1 are relevant to the current search.

This command returns a result code of kFPEOFErr only when it has reached the end of the volume directory tree. For example, if the client requests ten matches, the server may return only four matches, without returning an error. The client should then send a request for six (ten minus four) more matches, using the same CatalogPosition value that was received in the previous reply. This process continues until the originally requested matches are received or a result code of kFPEOFErr is returned. If this command returns a result code of kFPCatalogChanged, the client cannot continue the search. The client must restart the search by setting the first word of CatalogPosition to zero.

This command returns parameters for files, directories or both, depending on the value of the FileRsltBitmap and DirectoryRsltBitmap parameters. If FileRsltBitmap is null, this command assumes that you are not searching for files. Likewise, if DirectoryRsltBitmap is null, this command assumes that you are not searching for directories. If both parameters are non-zero, this command searches for files and directories. Note that if you are searching for both files and directories, certain restrictions apply with regard to the parameters that are searched. The rest of this section describes these restrictions.

The ReqBitmap parameter specifies the directory and file parameters to be searched. The low-order word of ReqBitmap is the same as low-order word of the File bitmap and the Directory bitmap in FPGetFileDirParms (page 80), with the exception of the Short Name parameter, which cannot be searched.

The high bit of the high-order word of `ReqBitmap` indicates whether the search should match on full names or partial names (0 = full name, 1 = partial name). There is no equivalent to the fsSBNegate bit used by the Macintosh File Manager's `PBCatSearch` function.

Figure 11 (page 32) shows parameters this command can search when it is searching directories only.

Figure 11 Parameters `FPCatSearchExt` searches when searching directories only

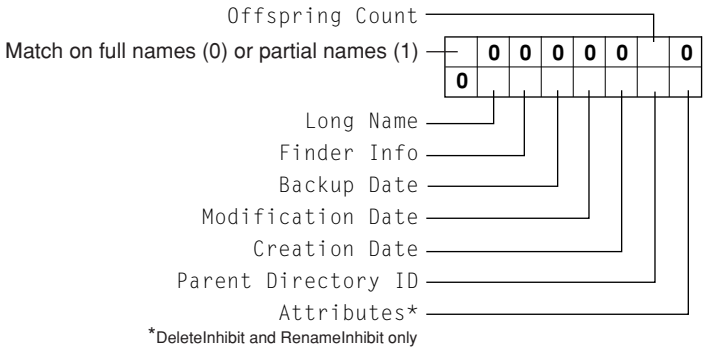


Figure 8 (page 27) shows parameters this command can search when it is searching files only.

Figure 12 Parameters `FPCatSearchExt` searches when searching files only

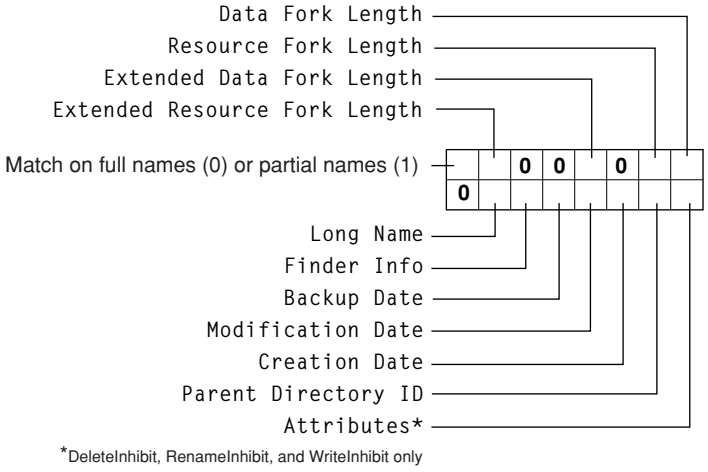
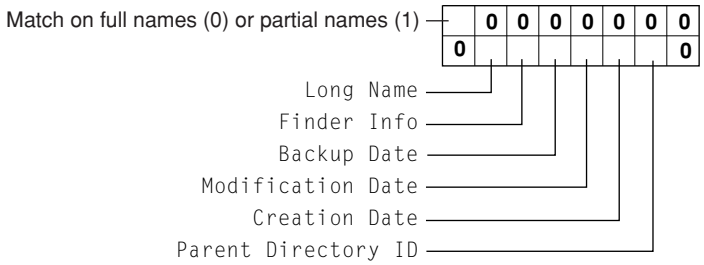


Figure 9 (page 27) shows parameters this command can search when it is searching both directories and files.

Figure 13 Parameters `FPCatSearchExt` searches when searching directories and files



Before sending this command, the user must call `FPOpenVol` (page 124) for the volume that is to be searched.

To return all files and directories that match the specified criteria, the user must have Read Only or Read & Write privileges for all directories. This command skips directories for which the user does not have Read Only or Read & Write privileges.

Table 8 (page 33) lists the result codes for the `FPCatSearchExt` command.

Table 8 Result codes for the `FPCatSearchExt` command

Result code	Explanation
`kFPCallNotSupported`	Server does not support this command.
`kFPCatalogChanged`	Catalog has changed and CatalogPosition may be invalid. No matches were returned.
`kFPEOFErr`	No more matches.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is null or bad.

Table 9 describes the reply block for the `FPCatSearchExt` command.

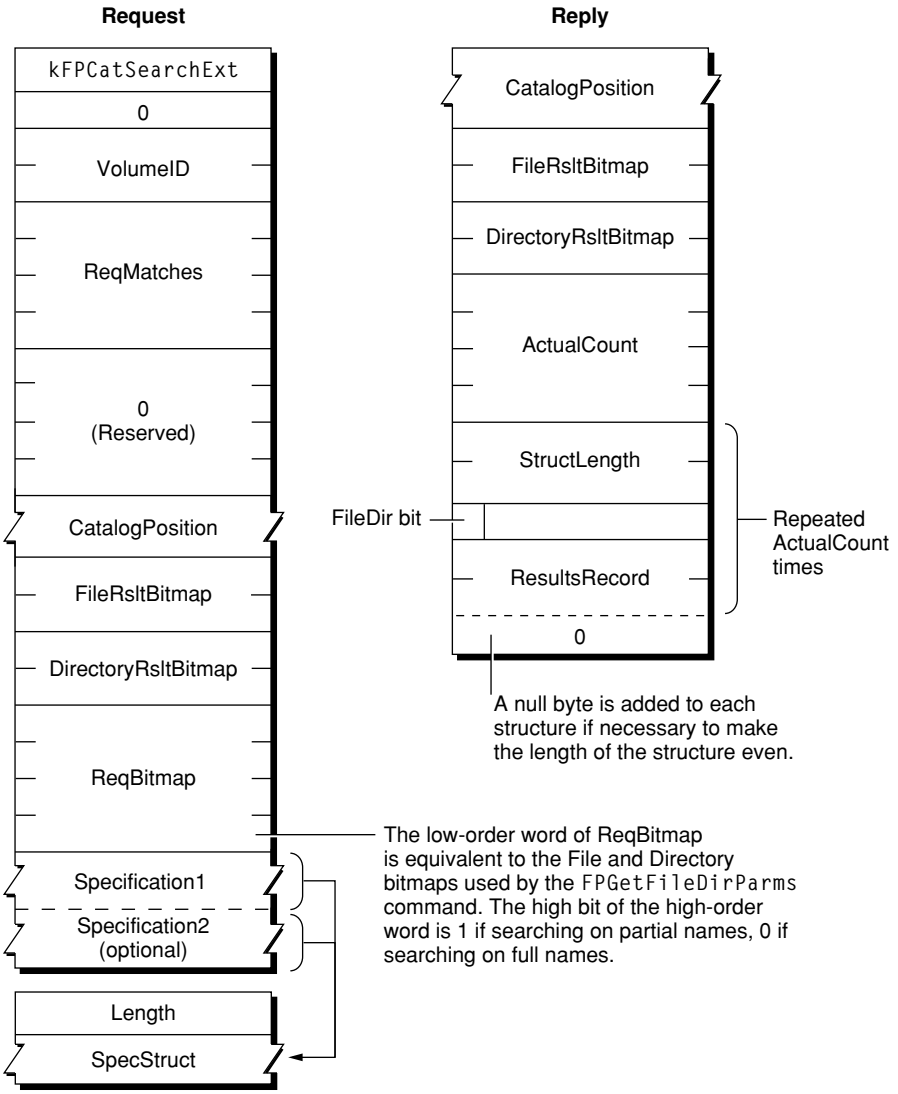
Table 9 Reply block for the `FPCatSearchExt` command

Name and size	Data
`CatalogPosition` (16 bytes)	Current position in the catalog.
`FileRsltBitmap` (short)	Copy of the input bitmap.
`DirectoryRsltBitmap` (short)	Copy of the input bitmap.
`ActualCount` (short)	Number of `ResultsRecord` structures that follow.

Name and size	Data
Zero or more `ResultsRecord` structures	Array of `ResultsRecord` structures describing the matches that were found and having the following structure: `StructLength` (byte) — Unsigned length of this structure including this byte and the byte for the `FileDir` bit. `FileDir` (bit 7 of a one-byte value) — Whether the record is for a file (0) or directory (1). `Results` — The matching Long Name, Parent Directory ID, or both with a trailing null byte if necessary to make the entire structure end on an even boundary.

Figure 14 shows the request and reply blocks for the `FPCatSearchExt` command.

Figure 14 Request and reply blocks for the `FPCatSearchExt` command



FPChangePassword

Allows users to change their passwords.

```
byte CommandCode
byte Pad
string UAM
string UserName
UserAuthInfo
```

Parameters

CommandCode

kFPChangePassword (36).

Pad

Pad byte.

UAM

String specifying the UAM to uses.

UserName

Name of the user whose password is to be changed. Starting with AFP 3.0, UserName is two bytes with each byte set to zero. The first byte indicates a zero length string, and the second byte is a pad byte.

UserAuthInfo

UAM-specific information.

Result

kFPNoErr if no error occurred. See Table 10 (page 36) for other possible result codes.

ReplyBlock

None.

Discussion

If the UAM is Cleartxt Passwrd, the AFP client sends the server the user's name plus the user's old and new eight-byte passwords in cleartext. The server looks up the password for that user. If it matches the old password sent in the packet, the new password is saved for that user. For more information on the Cleartext Password UAM, see the section "Cleartext Password" in the "Introduction" section.

If the UAM is Randnum Exchange, DES is used to encrypt and decrypt passwords. The AFP client sends the server the user name, the user's old eight-byte password encrypted with the user's new eight-byte password, and the user's new eight-byte password encrypted with the user's old eight-byte password. The server looks up the password for that user, uses that password as a key to decrypt the new password, and uses the result to decrypt the old password. If the final result matches what the server knows to be the old password, the new password is saved for that user. For more information on the Random Number Exchange UAM, see the section "Random Number Exchange" in the "Introduction" section.

When using the Random Number Exchange UAM, be sure to append null bytes to any password that is less than eight bytes so that the resulting password has a length of eight bytes.

If the user logged in using the Two-Way Random Number Exchange UAM, the client uses the Randnum UAM for changing the user's password.

If the UAM is DHCAST128, the AFP client must call FPChangePassword twice. The first time, the AFP client calls FPChangePassword to send the user name and a random number that has been encrypted. The server replies with an ID, a random number, and a nonce/server signature value encrypted by a session key. The AFP client calls FPChangePassword again, this time sending the user name and the ID returned by the

server. The client also sends the nonce incremented by one, the new password, and the old password, all encrypted by the session key. For information on using the DHX UAM to change passwords, see the section "DHX and Changing a Password" in the "Introduction" section.

Servers are not required to support this command. Call `FPGetSrvrInfo` (page 91) to determine whether a server supports this command.

The user may not have been granted the ability to change his or her password. Granting the ability to change a password is an administrative function and is beyond the scope of this protocol specification.

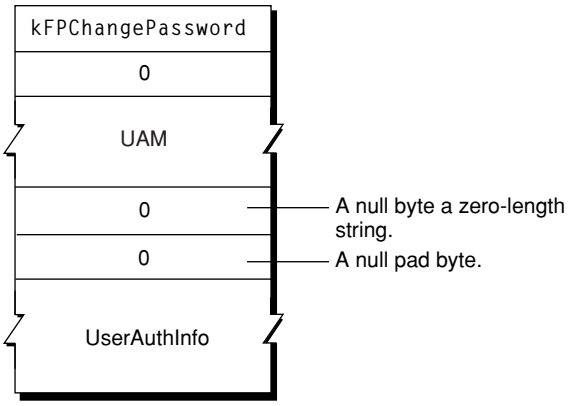
Table 10 lists the result codes for the `FPChangePassword` command.

Table 10 Result codes for the `FPChangePassword` command

Result code	Explanation
`kFPNoErr`	No error occurred.
`kFPCallNotSupported`	Server does not support this command.
`kFPUserNotAuth`	UAM failed (the specified old password doesn't match) or no user is logged in yet for the specified session.
`kFPBadUAM`	Specified UAM is not a UAM that `FPChangePassword` supports.
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPParamErr`	User name is null, exceeds the UAM's user name length limit, or does not exist.
`kFPPwdSameErr`	User attempted to change his or her password to the same password that he or she previously had. This error occurs only if the password expiration feature is enabled on the server.
`kFPPwdTooShortErr`	User password is shorter than the server's minimum password length, or user attempted to change password to a password that is shorter than the server's minimum password length.
`kFPPwdPolicyErr`	New password does not conform to the server's password policy.
`kFPMiscErr`	Non-AFP error occurred.

Figure 15 (page 37) shows the request block for the `FPChangePassword` command.

Figure 15 Request block for the `FPChangePassword` command



FPCloseDir

Closes a directory and invalidates its Directory ID.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
```

Parameters

CommandCode

`kFPCloseDir` (3).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Directory ID.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number, Volume ID, or Directory ID is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

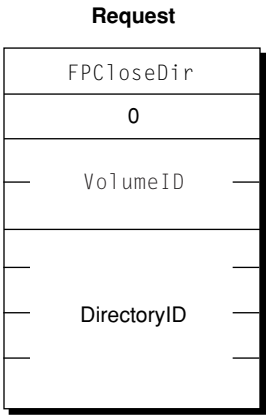
Discussion

This command invalidates the Directory ID specified by DirectoryID.

This command should be used only for variable Directory ID volumes. The user must have previously called `FPOpenVol` (page 124) for this volume and `FPOpenDir` (page 118) for this directory.

Figure 16 (page 38) shows the request block for the `FPCloseDir` command.

Figure 16 Request block for the `FPCloseDir` command



FPCloseDT

Closes a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
```

Parameters

CommandCode

`kFPCloseDT` (49).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or the Desktop database reference number is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

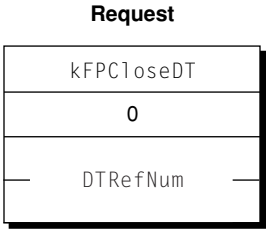
Discussion

This command invalidates the Desktop database reference number specified by `DTRefNum`.

The user must first have sent a successful `FPOpenDT` (page 120) command.

Figure 17 (page 39) shows the request block for the `FPCloseDT` command.

Figure 17 Request block for the `FPCloseDT` command



FPCloseFork

Closes a fork.

```
byte CommandCode
byte Pad
short OForkRefNum
```

Parameters

CommandCode

`kFPCloseFork` (4).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or the open fork reference number is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

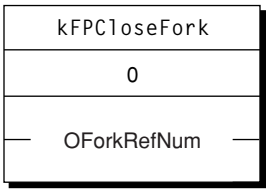
Discussion

This command causes the server to flush and close the specified fork, invalidating the open fork reference number. If the fork was written to, the file's modification date is set to the server's clock.

The user must first have sent a successful `FPOpenFork` (page 121) command.

Figure 18 (page 39) shows the request block for the `FPCloseFork` command.

Figure 18 Request block for the `FPCloseFork` command



FPCloseVol

Closes a volume.

```
byte CommandCode
byte Pad
short VolumeID
```

Parameters

CommandCode

`kFPCloseVol` (2).

Pad

Pad byte.

VolumeID

Volume ID.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or the Volume ID is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

Discussion

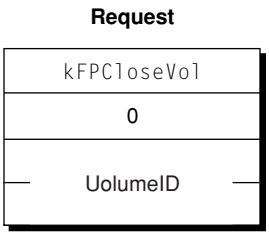
This command invalidates the specified Volume ID but does not necessarily close all open files on a volume before closing the volume, so you should close all open files before calling FPCloseVol.

The user must first have sent a successful `FPOpenVol` (page 124) command for this volume.

After sending this command, the user can send no other commands for this volume without opening the volume again.

Figure 19 (page 40) shows the request block for the `FPCloseVol` command.

Figure 19 Request block for the `FPCloseVol` command



FPCopyFile

Copies a file from one location to another on the same file server.

```
byte CommandCode
byte Pad
short SourceVolumeID
long SourceDirectoryID
short DestVolumeID
long DestDirectoryID
byte SourcePathType
string SourcePathname
byte DestPathType
string DestPathname
byte NewType
string NewName
```

Parameters

CommandCode

`kFPCopyFile` (5).

Pad

Pad byte.

SourceVolumeID

Source Volume ID.

SourceDirectoryID

Source ancestor Directory ID.

DestVolumeID

Destination Volume ID.

DestDirectoryID

Destination ancestor Directory ID.

SourcePathType

Type of names in `SourcePathname`. See Path Type Constants (page 174) for possible values.

SourcePathname

Pathname of the file to be copied (cannot be null). `SourcePathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

DestPathType

Type of names in `DestPathname`. See Path Type Constants (page 174) for possible values.

DestPathname

Pathname to the destination parent directory (may be null). `DestPathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

NewType

Type of name in `NewName`. See Path Type Constants (page 174) for possible values.

NewName

Name to be given to the copy (may be null).

Result

`kFPNoErr` if no error occurred. See Table 11 (page 42) for other possible result codes.

ReplyBlock

None.

Discussion

This command copies a file to a new location on the server. The source and destination can be on the same or on different volumes.

The server tries to open the source file for Read, DenyWrite access. If this fails, the server returns `kFPDenyConflict` as the result code. If the server successfully opens the file, it copies the file to the directory specified by the destination parameters.

The copy is given the name specified by the `NewName` parameter. If `NewName` is null, the server gives the copy the same name as the original. The file's other name (Long, Short) is generated as described in the section "Catalog Node Names" in Chapter 1. A unique file number is assigned to the file. The server also sets the file's Parent ID to the Directory ID of the destination parent directory. All other file parameters remain the same as the source file's parameters. The modification date of the destination parent directory is set to the server's lock.

The user must have search access to all ancestors of the source file, except the source parent directory, and read access to the source parent directory. Further, the user must have search or write access to all ancestors of the destination file, except the destination parent directory, and write access to the destination parent directory.

This command is optional and may not be supported by all servers.

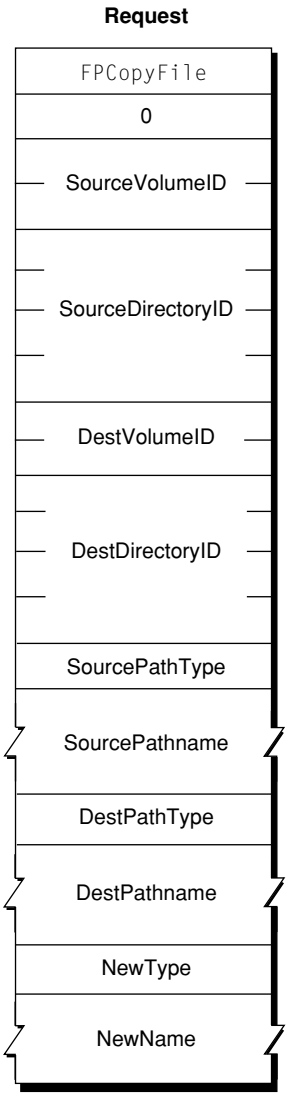
Table 11 lists the result codes for the `FPCopyFile` command.

Table 11 Result codes for the `FPCopyFile` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to read the file or write to the destination.
`kFPCallNotSupported`	Server does not support this command.
`kFPDenyConflict`	File cannot be opened for Read, DenyWrite.
`kFPDiskFull`	No more space exists on the destination volume.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectExists`	File or directory of the name specified by `NewName` already exists in the destination parent directory.
`kFPObjectNotFound`	The source file does not exist; ancestor directory is unknown.
`kFPObjectTypeErr`	Source parameters point to a directory.
`kFPParamErr`	Open fork reference number is unknown; a combination of the `StartEndFlag` bit and `Offset` parameters specifies a range that starts before byte zero.

Figure 20 (page 43) shows the request block for the `FPCopyFile` command.

Figure 20 Request block for the `FPCopyFile` command



FPCreateDir

Creates a new directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPCreateDir` (6).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname, including the name of the new directory (cannot be null). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8– encoded path.

Result

`kFPNoErr` if no error occurred. See Table 12 (page 44) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a long, called `NewDirectoryID`, containing the Directory ID of the new directory in the reply block.

Discussion

This command creates an empty directory having the name specified by the `Pathname` parameter. The file server assigns the directory a unique Directory ID and returns it in the reply block. The new directory's Owner ID is set to the User ID of the user sending the command, and its Group ID is set to the ID of the user's Primary Group ID, if a primary group has been specified for the user.

The new directory's privileges are initially set to read, write, and search for the owner, with no privileges for a group or Everyone. Finder information is set to zero and all directory attributes are initially cleared. The directory's creation and modification dates, as well as the modification date of the parent directory, are set to the server's clock. The directory's backup date is set to $80000000, signifying that the directory has never been backed up. The directory's other names are generated as described in the section "Catalog Node Names" in Chapter 1.

The user must have search or write access to all ancestors, except this directory's parent directory, as well as write access to the parent directory.

Table 12 lists the result codes for the `FPCreateDir` command.

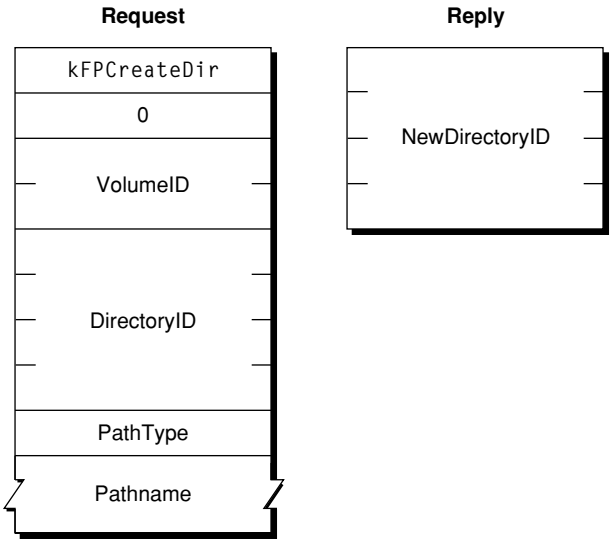
Table 12 Result codes for the `FPCreateDir` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPDiskFull`	No more space exists on the volume.
`kFPFlatVol`	Volume is flat and does not support directories.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Ancestor directory is unknown.
`kFPObjectExists`	File or directory of the specified name already exists.
`kFPParamErr`	Session reference number, Volume ID, or pathname is null or invalid.

Result code	Explanation
kFPVolLocked	Destination volume is read-only.

Figure 21 (page 45) shows the request and reply blocks for the FPCreateDir command.

Figure 21 Request and reply blocks for the FPCreateDir command



FPCreateFile

Creates a new file.

```
byte CommandCode
byte Flag
short VolumeID
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

kFPCreateFile (7).

Flag

Bit 7 of the Flag parameter is the CreateFlag bit, where 0 indicates a soft create and 1 indicates a hard create.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

PathType

Type of names in Pathname. See Path Type Constants (page 174) for possible values.

Pathname

Pathname, including the name of the new file (cannot be null). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 13 (page 46) for other possible result codes.

ReplyBlock

None.

Discussion

This command creates an empty file having the name specified by `Pathname`. For a soft create, if a file by that name already exists, the server returns a result code of `kFPObjectExists`. Otherwise, it creates a new file and assigns it the name specified by `Pathname`. A unique file number is assigned to the file. Finder information is set to zero, and all file attributes are initially cleared. The file's creation and modification dates, and the modification date of the file's parent of the file's parent directory, are set to the server's clock. The file's backup date is set to $80000000, signifying that this file has never been backed up. The file's other names are generated as described in the section "Catalog Node Names" in Chapter 1. The lengths of both of the file's forks are set to zero.

For a soft create, the user must have search or write access to all ancestors, except this file's parent directory, as well as write access to the parent directory. For a hard create, the user must have search access to all ancestors, except the parent directory, as well as read and write access to the parent directory.

For a hard create, if the file already exists and is not open, the file is deleted and then recreated. All file parameters (including the creation date) are reinitialized as described above.

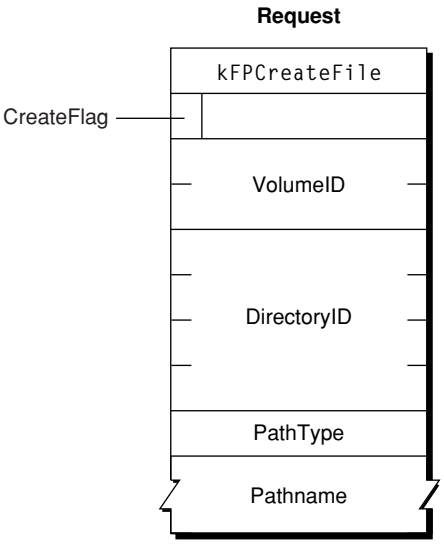
Table 13 lists the result codes for the `FPCreateFile` command.

Table 13 Result codes for the `FPCreateFile` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPDiskFull`	No more space exists on the volume.
`kFPFileBusy`	If attempting a hard create, the file already exists and is open.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectExists`	If attempting a soft create, a file of the specified name already exists.
`kFPObjectNotFound`	Ancestor directory is unknown.
`kFPVolLocked`	Destination volume is read-only.
`kFPParamErr`	Session reference number, Volume ID, or pathname is null or invalid.

Figure 22 (page 47) shows the request block for the `FPCreateFile` command.

Figure 22 Request block for the `FPCreateFile` command



FPCreateID

Creates a unique File ID for a file.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPCreateID` (39).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Directory ID of the directory in which the file is to be created.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Name of the file that is the target of the File ID (that is, the filename of the file for which a File ID is being created). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 14 (page 48) for other possible result codes.

ReplyBlock

None.

Discussion

File IDs provide a way to keep track of a file even if its name or location changes. The scope of a File ID is limited to the files on a volume. File IDs cannot be used across volumes.

The AFP server should take steps to ensure that every File ID is unique and that no File ID is reused once it has been deleted.

The user must have the Read Only or the Read & Write privilege to use this command.

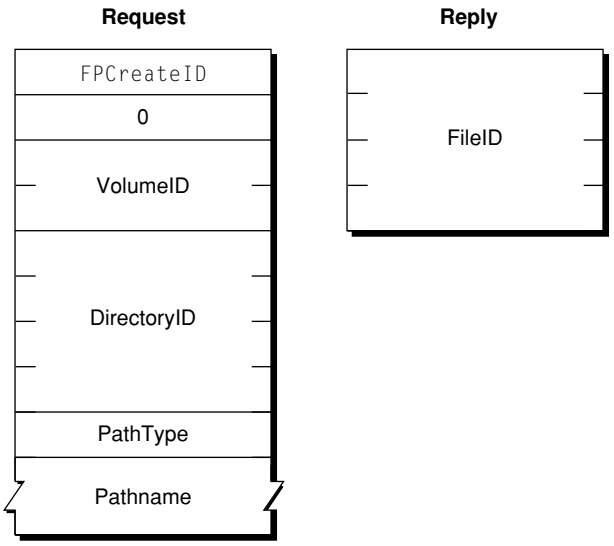
Table 14 lists the result codes for the `FPCreateID` command.

Table 14 Result codes for the `FPCreateID` command

Result code	Explanation
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Target file does not exist.
`kFPObjectTypeErr`	Object defined was a directory, not a file.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is null or bad.
`kFPVolLocked`	Destination volume is read-only.

Figure 23 (page 48) shows the request block for the `FPCreateID` command.

Figure 23 Request block for the `FPCreateID` command



FPDelete

Deletes a file or directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

kFPDelete (8).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

PathType

Type of names in Pathname. See Path Type Constants (page 174) for possible values.

Pathname

Pathname of the file or directory to be deleted (may be null if a directory is to be deleted). Pathnameis a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

kFPNoErr if no error occurred. See Table 15 (page 49) for other possible result codes.

ReplyBlock

None.

Discussion

When deleting a directory, the server checks to see if it contains any offspring. If a directory contains offspring, the server returns a result code of kFPDirNotEmpty. If a file that is to be deleted is open by any user, the server returns a result code of kFPFileBusy. The modification date of the parent directory of the deleted file or directory is set to the servers clock.

The user must have search access to all ancestors except the file or directory's parent directory, as well as write access to the parent directory. If a directory is being deleted, the user must also have search access to the parent directory; for a file, the user must also have read access to the parent directory.

The AFP server identifies the Network Trash Folder by name, and that name is not localized in international versions of the Mac OS because it is invisible.

Table 15 lists the result codes for the FPDelete command.

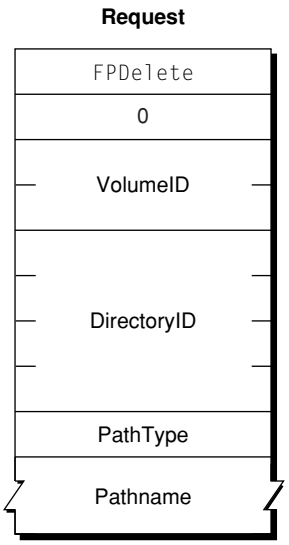
Table 15 Result codes for the FPDelete command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.

Result code	Explanation
kFPDirNotEmpty	Directory is not empty.
kFPMiscErr	Non-AFP error occurred.
kFPObjectLocked	File or directory is marked DeleteInhibit.
kFPObjectNotFound	Input parameters do not point to an existing file or directory.
kFPObjectTypeErr	Object defined was a directory, not a file.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown; pathname is invalid.
kFPVolLocked	Volume is read-only.

Figure 24 (page 50) shows the request block for the `FPDelete` command.

Figure 24 Request block for the `FPDelete` command



FPDeleteID

Invalidates all instances of the specified File ID.

```
byte CommandCode
byte Pad
short VolumeID
long FileID
```

Parameters

CommandCode

`kFPDeleteID` (40).

Pad
Pad byte.

VolumeID
Volume ID.

FileID
File ID that is to be deleted.

Result
`kFPNoErr` if no error occurred. See Table 16 for other possible result codes.

ReplyBlock
None.

Discussion
This command deletes the specified File ID, which was created by an earlier call to `FPCreateID` (page 47).

The user must have the Read Only or the Read & Write access privilege to use this command.

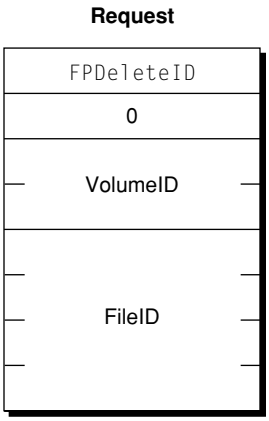
Table 16 lists the result codes for the `FPDeleteID` command.

Table 16 Result codes for the `FPDeleteID` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPCallNotSupported`	Server does not support this command.
`kFPIDNotFound`	File ID was not found. (No file thread exists.)
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Target file does not exist. The File ID is deleted anyway.
`kFPObjectTypeErr`	Object defined was a directory, not a file.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is null or bad.
`kFPVolLocked`	Volume is read-only.

Figure 25 (page 52) shows the request block for the `FPDeleteID` command.

Figure 25 Request block for the `FPDeleteID` command



FPDisconnectOldSession

Disconnects an old session and transfers its resources to a new session.

```
byte CommandCode
byte Pad
short Type
long TokenLength
string Token
```

Parameters

CommandCode

`kFPDisconnectOldSession` (65).

Pad

Pad byte.

Type

Volume ID.

TokenLength

Length of `Token`.

Token

Token previous obtained by calling `FPGetSessionToken` (page 89).

Result

`kFPNoErr` if no error occurred, `kFPCallNotSupported` if the server does not support this command, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

Discussion

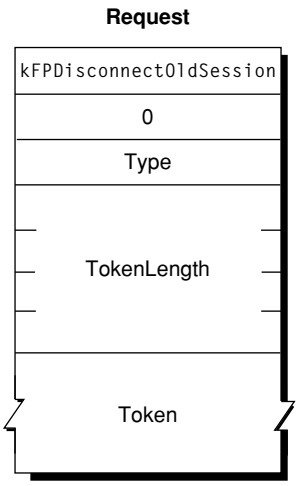
This command disconnects the session identified by the Token parameter, which was obtained by previously calling `FPGetSessionToken` (page 89) and transfers the resources of the old session to the new session.

The AFP client calls this command when the session it previously established was inadvertently disconnected, it successfully establishes a new session, and it is able to restore the state of the previous session. If the AFP client cannot successfully reestablish the state of the previous session, it should call this command, log out, and report the failure to the local operating system.

If the AFP client successfully reestablishes the state of the previous session, it should call this command again to get a new session token.

Figure 26 (page 53) shows the request block for the `FPDisconnectOldSession` command.

Figure 26 Request block for the `FPDisconnectOldSession` command



FPEnumerate

Lists the contents of a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short FileBitmap
short DirectoryBitmap
short ReqCount
short StartIndex
short MaxReplySize
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPEnumerate` (9).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID
Identifier for the directory to list.

FileBitmap
Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see File Bitmap (page 164).

DirectoryBitmap
Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see Directory Bitmap (page 162).

ReqCount
Maximum number of `ResultsRecord` structures for which information is to be returned.

StartIndex
Directory offspring index.

MaxReplySize
Maximum size of the reply block.

PathType
Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname
Pathname to the desired directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result
`kFPNoErr` if no error occurred. See Table 17 (page 55) for other possible result codes.

ReplyBlock
If the result code is `kFPNoErr`, the server returns a reply block. See Table 18 (page 56) for the format of the reply block.

Discussion
This command enumerates a directory as specified by the input parameters. This command differs from the `FPEnumerateExt` (page 57) and `FPEnumerateExt2` (page 61) commands in that it is not able to handle values that may be returned when volumes are larger than 4 GB in size.

If `FileBitmap` is null, only directory offspring are enumerated, and `StartIndex` can range from one to the total number of directory offspring. Similarly, if `DirectoryBitmap` is null, only file offspring are enumerated, and `StartIndex` can range from one to the total number of file offspring. If both bitmaps have bits set, `StartIndex` can range from one to the total number of offspring. In this case, offspring structures for both files and directories are returned. These structures are not returned in any particular order.

This command is completed when the number of structures specified by `ReqCount` has been inserted into the reply block, when the reply block is full, or when no more offspring exist to be enumerated. No partial offspring structures are returned.

The server retrieves the specified parameters for each enumerated offspring and packs them, in bitmap order, in structures in the reply block. The server inserts one copy of the input bitmaps before all of the structures.

The server needs to keep variable-length parameters, such as Long Name and Short Name, at the end of each structure. In order to do this, the server represents variable-length parameters in the bitmap order as fixed-length offsets (integers). Each offset is measured from the start of the parameters in each structure (not from the start of the bitmap or the start of the header bytes) to the start of the variable-length parameter. Each structure is padded (suffixed) with a null byte if necessary to make its length even.

If this command returns a result code of `kFPNoErr`, all structures in the reply block are valid. If any error result code is returned, no valid offspring structures are in the reply block.

If the OffSpring Count bit in the Directory bitmap is set, the server adjusts the Offspring Count of each directory to reflect the access rights the user has to that directory. For example, if a particular directory contains three file and two directory offspring, the server returns the Offspring Count as 2 if the user has only search access to the directory, 3 if the user has only read access to the directory, or 5 if the user has both search and read access to the directory.

The user must have search access to all ancestors except this directory. In addition, the user needs search access to this directory in order to enumerate directory offspring and read access in order to enumerate file offspring.

Enumerating a large directory may require the sending of several `FPEnumerate` commands. During that time, other users may add to or delete from the directory, so enumeration can miss offspring or return duplicate offspring. To enumerate a directory accurately, enumerate until a `kFPObjectNotFound` result code is returned and then filter out duplicate entries.

A given offspring is not guaranteed to occupy the same index number in the parent directory from one enumeration to the next.

Table 17 lists the result codes for the `FPEnumerate` command.

Table 17 Result codes for the `FPEnumerate` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to retrieve a parameter that cannot be retrieved by this command, an attempt was made to retrieve the Directory ID for a directory on a variable Directory ID volume, or both bitmaps are empty.
`kFPDirNotFound`	Input parameters do not point to an existing directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	No more offspring exist to be enumerated.
`kFPObjectTypeErr`	Input parameters point to a file.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown, pathname is bad, or `MaxReplySize` is too small to hold a single offspring structure.

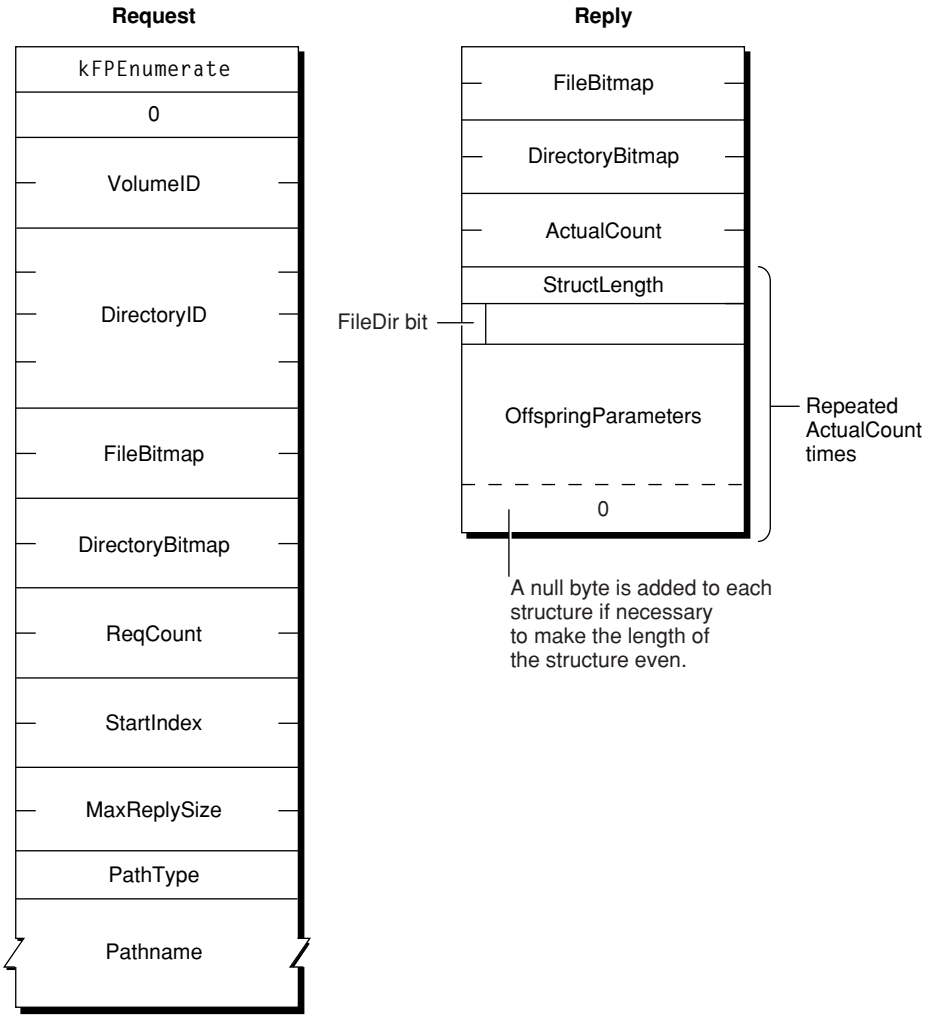
Table 18 describes the reply block for the `FPEnumerate`command.

Table 18 Reply block for the `FPEnumerate` command

Name and size	Data
`FileBitmap` (short)	Copy of the input parameter.
`DirectoryBitmap` (short)	Copy of the input parameter.
`ActualCount` (short)	Actual number of `ResultsRecord` structures returned.
Zero or more `ResultsRecord` structures	Array of `ResultsRecord` structures consisting of the following fields: `StructLength` (byte) — Unsigned length of this structure, including this byte and the byte for the `FileDir` bit. `FileDir` (bit) — Flag indicating whether the `OffspringParameters` field describes a file (0) or a directory (1). `OffspringParameters` — Packed in bitmap order, with a trailing null byte if necessary to make the length of the entire structure an even number.

Figure 27 shows the request and reply blocks for the `FPEnumerate` command.

Figure 27 Request and reply blocks for the `FPEnumerate` command



FPEnumerateExt

Lists the contents of a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short FileBitmap
short DirectoryBitmap
short ReqCount
short StartIndex
short MaxReplySize
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPEnumerateExt` (66).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Identifier for the directory to list.

FileBitmap

Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see File Bitmap (page 164).

DirectoryBitmap

Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see Directory Bitmap (page 162).

ReqCount

Maximum number of `ResultsRecord` structures for which information is to be returned.

StartIndex

Directory offspring index.

MaxReplySize

Maximum size of the reply block.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 19 (page 59) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 20 (page 60) for the format of the reply block.

Discussion
This command enumerates a directory as specified by the input parameters. This command differs from the `FPEnumerate` (page 53) command in that this command is prepared to handle values that may be returned when volumes are larger than 4 GB in size. This command also differs from the `FPEnumerateExt2` (page 61) command in that StartIndex and MaxReplySize are shorts (instead of longs), which may limit the number of entries in a single directory that can be listed. The reply block for this command is the same as the reply block for `FPEnumerateExt2`.

If `FileBitmap` is null, only directory offspring are enumerated, and `StartIndex` can range from one to the total number of directory offspring. Similarly, if the `DirectoryBitmap` is null, only file offspring are enumerated, and `StartIndex` can range from one to the total number of file offspring. If both bitmaps have bits set, `StartIndex` can range from one to the total number of offspring. In this case, offspring structures for both files and directories are returned. These structures are not returned in any particular order.

This command is completed when the number of structures specified by `ReqCount` has been inserted into the reply block, when the reply block is full, or when no more offspring exist to be enumerated. No partial offspring structures are returned.

The server retrieves the specified parameters for each enumerated offspring and packs them, in bitmap order, in structures in the reply block. The server inserts one copy of the input bitmaps before all of the structures.

The server needs to keep variable-length parameters, such as Long Name and Short Name, at the end of each structure. In order to do this, the server represents variable-length parameters in the bitmap order as fixed-length offsets (integers). Each offset is measured from the start of the parameters in each structure (not from the start of the bitmap or the start of the header bytes) to the start of the variable-length parameter. Each structure will be padded (suffixed) with a null byte if necessary to make its length even.

If `kFPNoErr` is returned, all structures in the reply block are valid. If any error result code is returned, no valid offspring structures exist in the reply block.

If the OffSpring Count bit in the Directory bitmap is set, the server adjusts the Offspring Count of each directory to reflect what access rights the user has to that directory. For example, if a particular directory contains three file and two directory offspring, the server will return its Offspring Count as 2 if the user has only search access to the directory, 3 if the user has only read access to the directory, or 5 if the user has both search and read access to the directory.

The user must have search access to all ancestors except this directory. In addition, the user needs search access to this directory in order to enumerate directory offspring and read access in order to enumerate file offspring.

Enumerating a large directory may require the sending of several `FPEnumerateExt` commands. During that time, other users may add to or delete from the directory, so enumeration can miss offspring or return duplicate offspring. To enumerate a directory accurately, enumerate until a `kFPObjectNotFound` result code is returned and then filter out duplicate entries.

A given offspring is not guaranteed to occupy the same index number in the parent directory from one enumeration to the next.

Table 19 lists the result codes for the `FPEnumerateExt` command.

Table 19 Result codes for the `FPEnumerateExt` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.

Result code	Explanation
kFPBitmapErr	Attempt was made to retrieve a parameter that cannot be retrieved by this command, an attempt was made to retrieve the Directory ID for a directory on a variable Directory ID volume, or both bitmaps are empty.
kFPDirNotFound	Input parameters do not point to an existing directory.
kFPMiscErr	Non-AFP error occurred.
kFPObjectNotFound	No more offspring exist to be enumerated.
kFPObjectTypeErr	Input parameters point to a file.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown, pathname is bad, or MaxReplySize is too small to hold a single offspring structure.

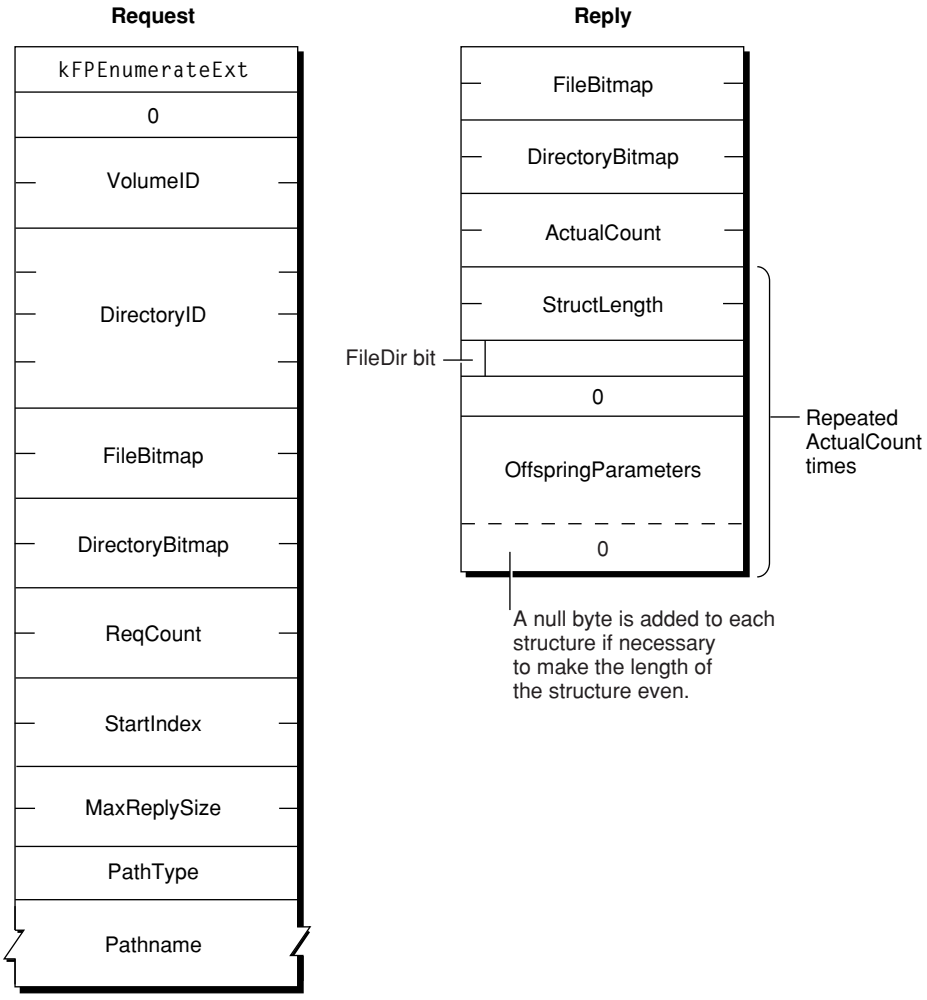
Table 20 describes the reply block for the FPEnumerateExt command.

Table 20 Reply block for the FPEnumerateExt command

Name and size	Data
DirectoryBitmap (short)	Copy of the input parameter.
ActualCount (short)	Actual number of ResultsRecord structures returned.
Zero or more ResultsRecord structures	An array of ResultsRecord structure consisting of the following fields: StructLength (byte) — Unsigned length of this structure, including this byte and the byte for the FileDir bit. FileDir (bit) — Flag indicating whether the OffspringParameters field describes a file (0) or a directory (1). Pad (byte) — Always zero. OffspringParameters — Packed in bitmap order, with a trailing null byte if necessary to make the length of the entire structure an even number.

Figure 28 shows the request and reply blocks for the FPEnumerateExt command.

Figure 28 Request and reply blocks for the `FPEnumerateExt` command



FPEnumerateExt2

Lists the contents of a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short FileBitmap
short DirectoryBitmap
short ReqCount
long StartIndex
long MaxReplySize
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPEnumerateExt2` (68).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Identifier for the directory to list.

FileBitmap

Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see File Bitmap (page 164).

DirectoryBitmap

Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for this bitmap, see Directory Bitmap (page 162).

ReqCount

Maximum number of `ResultsRecord` structures for which information is to be returned.

StartIndex

Directory offspring index.

StartIndex

Maximum size of the reply block.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 21 (page 63) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 22 (page 64) for the format of the reply block.

Discussion
This command enumerates a directory as specified by the input parameters. This command differs from the `FPEnumerateExt` (page 57) command in that `StartIndex` and `MaxReplySize` are longs instead of shorts, thereby allowing this command to list more entries in a single directory. The reply block for this command is the same as the reply block for the `FPEnumerateExt` command.

If `FileBitmap` is null, only directory offspring are enumerated, and `StartIndex` can range from one to the total number of directory offspring. Similarly, if the `DirectoryBitmap` is null, only file offspring are enumerated, and `StartIndex` can range from one to the total number of file offspring. If both bitmaps have bits set, `StartIndex` can range from one to the total number of offspring. In this case, offspring structures for both files and directories are returned. These structures are not returned in any particular order.

This command is completed when the number of structures specified by `ReqCount` has been inserted into the reply block, when the reply block is full, or when no more offspring exist to be enumerated. No partial offspring structures are returned.

The server retrieves the specified parameters for each enumerated offspring and packs them, in bitmap order, in structures in the reply block. The server inserts one copy of the input bitmaps before all of the structures.

The server needs to keep variable-length parameters, such as Long Name and Short Name, at the end of each structure. In order to do this, the server represents variable-length parameters in the bitmap order as fixed-length offsets (integers). Each offset is measured from the start of the parameters in each structure (not from the start of the bitmap or the start of the header bytes) to the start of the variable-length parameter. Each structure will be padded (suffixed) with a null byte if necessary to make its length even.

If `kFPNoErr` is returned, all structures in the reply block are valid. If any error result code is returned, no valid offspring structures exist in the reply block.

If the OffSpring Count bit in the Directory bitmap is set, the server adjusts the Offspring Count of each directory to reflect what access rights the user has to that directory. For example, if a particular directory contains three file and two directory offspring, the server will return its Offspring Count as 2 if the user has only search access to the directory, 3 if the user has only read access to the directory, or 5 if the user has both search and read access to the directory.

The user must have search access to all ancestors except this directory. In addition, the user needs search access to this directory in order to enumerate directory offspring and read access in order to enumerate file offspring.

Enumerating a large directory may require the sending of several `FPEnumerateExt2` commands. During that time, other users may add to or delete from the directory, so enumeration can miss offspring or return duplicate offspring. To enumerate a directory accurately, enumerate until a `kFPObjectNotFound` result code is returned and then filter out duplicate entries.

A given offspring is not guaranteed to occupy the same index number in the parent directory from one enumeration to the next.

Table 21 lists the result codes for the `FPEnumerateExt2` command.

Table 21 Result codes for the `FPEnumerateExt2` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.

Result code	Explanation
kFPBitmapErr	Attempt was made to retrieve a parameter that cannot be retrieved by this command, an attempt was made to retrieve the Directory ID for a directory on a variable Directory ID volume, or both bitmaps are empty.
kFPDirNotFound	Input parameters do not point to an existing directory.
kFPMiscErr	Non-AFP error occurred.
kFPObjectNotFound	No more offspring exist to be enumerated.
kFPObjectTypeErr	Input parameters point to a file.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown, pathname is bad, or MaxReplySize is too small to hold a single offspring structure.

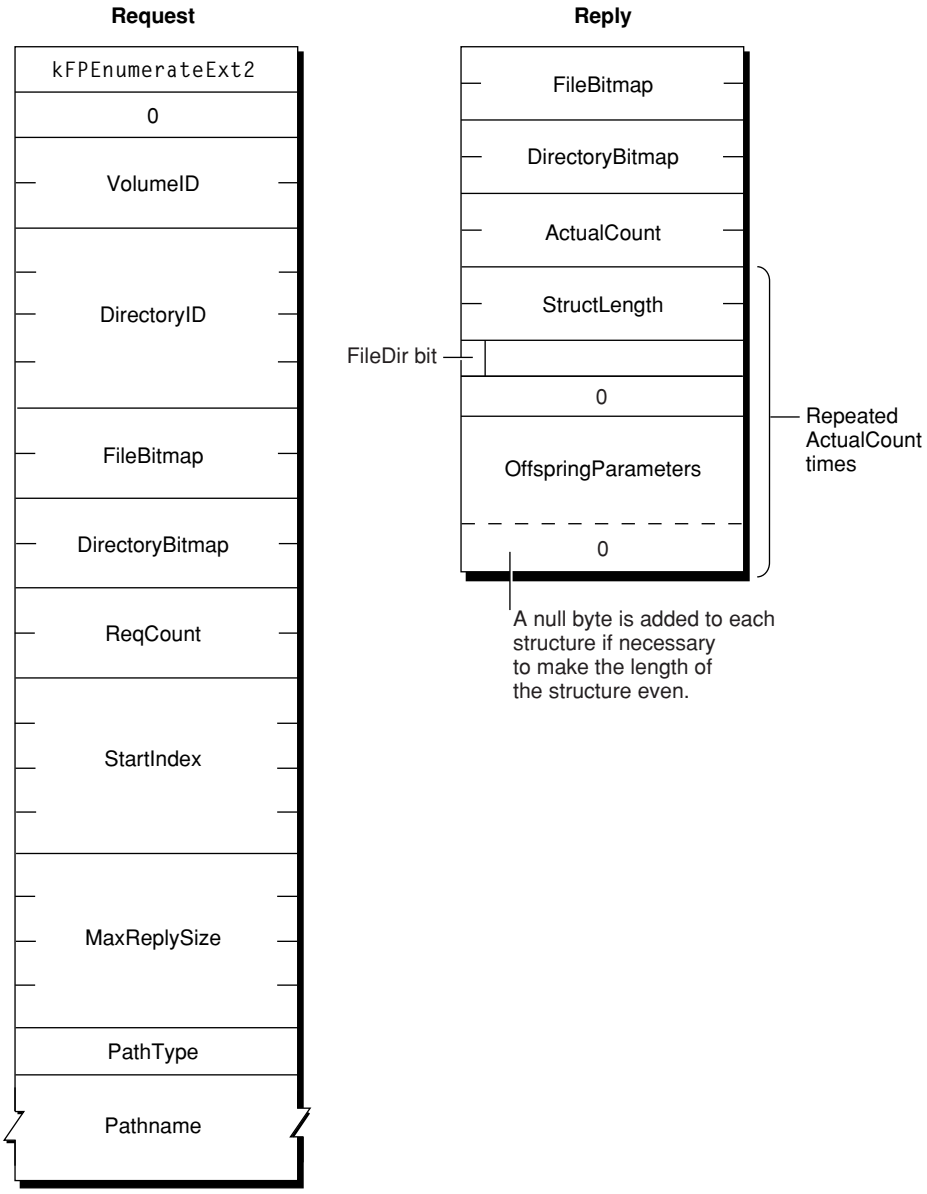
Table 22 describes the reply block for the FPEnumerateExt2 command.

Table 22 Reply block for the FPEnumerateExt2 command

Name and size	Data
FileBitmap (short)	Copy of the input parameter.
DirectoryBitmap (short)	Copy of the input parameter.
ActualCount (short)	Actual number of ResultsRecord structures returned.
Zero or more ResultsRecord structures	An array of ResultsRecord structures, each consisting of the following fields: StructLength (byte) — Unsigned length of this structure, including this byte and the byte for the FileDir bit. FileDir (bit) — Flag indicating whether the OffspringParameters field describes a file (0) or a directory (1). Pad (byte) — Always zero. OffspringParameters — Packed in bitmap order, with a trailing null byte if necessary to make the length of the entire structure an even number.

Figure 29 shows the request and reply blocks for the FPEnumerateExt2 command.

Figure 29 Request and reply blocks for the `FPEnumerateExt2` command



FPExchangeFiles

Preserves existing File IDs when performing a Save or a Save As operation.

```
byte CommandCode
byte Pad
short VolumeID
long SourceDirectoryID
long DestDirectoryID
byte SourcePathType
string SourcePathname
byte DestPathType
string DestPathname
```

Parameters

CommandCode

`kFPExchangeFiles` (42).

Pad

Pad byte.

VolumeID

Volume ID.

SourceDirectoryID

Identifier of the directory containing the source file.

DestDirectoryID

Identifier of the directory containing the destination file.

SourcePathType

Type of names in `SourcePathname`. See Path Type Constants (page 174) for possible values.

SourcePathname

Pathname of the source file. `SourcePathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

DestPathType

Type of names in `DestPathname`. See Path Type Constants (page 174) for possible values.

DestPathname

Pathname of the source file. `DestPathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 23 (page 67) for other possible result codes.

ReplyBlock

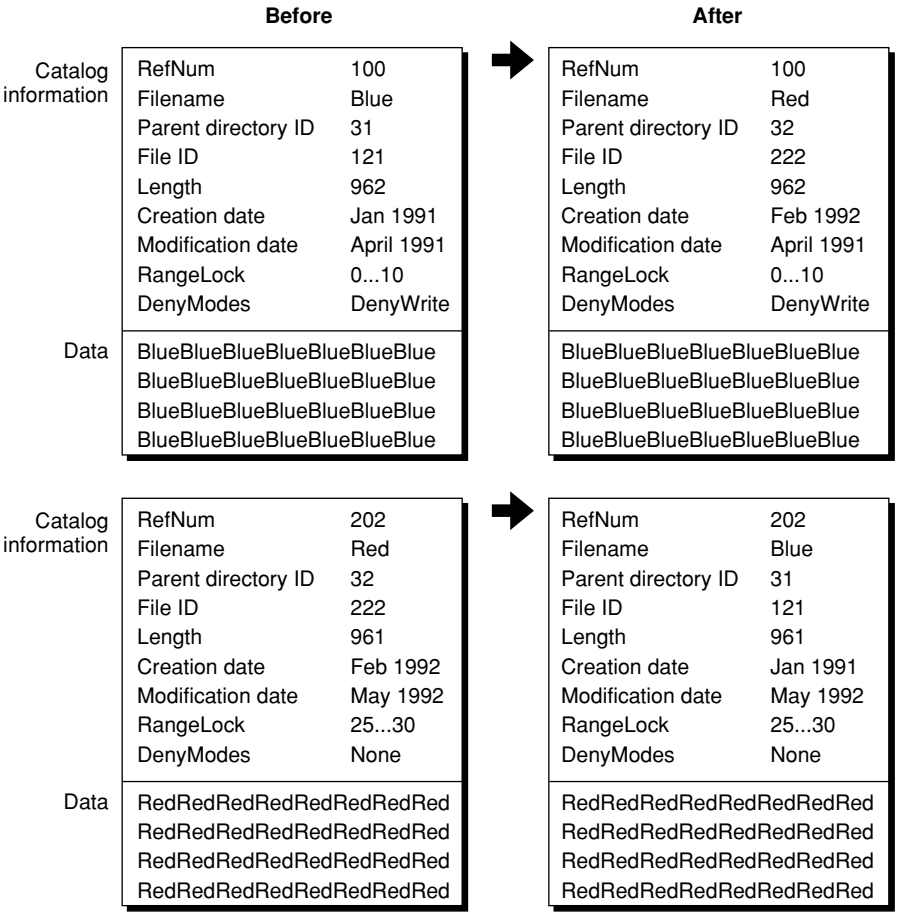
None.

Discussion

To use this command, both files must exist on the same volume. File IDs do not, however, have to exist on the files to be exchanged. The files being exchanged can be open or closed.

Figure 30 (page 67) shows the results of an exchange operation between two files named Blue and Red.

Figure 30 Example of exchanging files



Notice that only the filename, Parent Directory ID, File ID, and creation dates are exchanged. Byte-range locks and deny modes still apply to the same file reference number and data.

The user must have the Read & Write privilege for both files in order to use this command.

Table 23 lists the result codes for the `FPExchangeFiles` command.

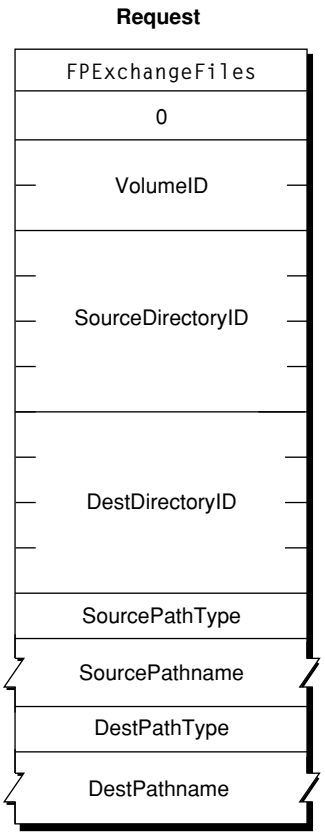
Table 23 Result codes for the `FPExchangeFiles` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBadIDErr`	File ID is not valid.
`kFPCallNotSupported`	Server does not support this command.
`kFPIDNotFound`	File ID was not found. (No file thread exists.)
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectTypeErr`	Object defined was a directory, not a file.

Result code	Explanation
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown; pathname is null or bad.

Figure 31 shows the request block for the FPExchangeFiles command.

Figure 31 Request block for the FPExchangeFiles command



FPFlush

Writes any volume data that has been modified.

```
byte CommandCode
byte Pad
short VolumeID
```

Parameters

CommandCode

kFPFlush (10).

Pad

Pad byte.

VolumeID
Volume ID.

Result
kFPNoErr if no error occurred, `kFPParamErr` if the session reference number or Volume ID is invalid, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock
None.

Discussion
This command writes to disk as much changed information as possible including

- all forks opened by the user
- volume catalog information changed by the user
- any updated volume data structures

AFP does not specify that the server must perform all of these functions. Therefore, users should not rely on the server to perform any particular function.

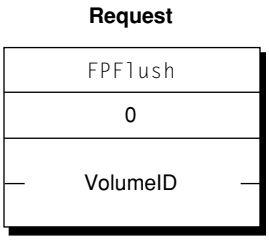
The volume's modification date may change as a result of this command, but uses should not rely on it; updating of the date is implementation-dependent. If no volume information was changed since the last `FPFlush` command, the date may or may not change.

Notice that only the filename, Parent Directory ID, File ID, and creation dates are exchanged. Byte-range locks and deny modes still apply to the same file reference number and data.

The user must have the Read & Write privilege for both files in order to use this command.

Figure 32 shows the request block for the `FPFlush` command.

Figure 32 Request block for the `FPFlush` command



FPFlushFork

Writes any data buffered from previous write commands.

```
byte CommandCode
byte Pad
short OForkRefNum
```

Parameters

CommandCode
`kFPFlushFork` (11).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or Volume ID is invalid, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

None.

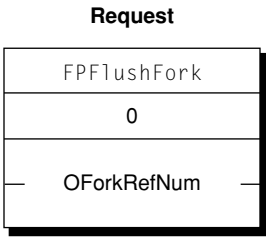
Discussion

This command writes to disk any data buffered by the server by previous write commands. If the fork has been modified, the server set's the file's modification date to the server's clock.

In order to optimize disk access, the server may buffer write commands made to a particular file fork. Within the constraints of performance, the server flushes each fork as soon as possible. By sending this command, the AFP client can force the server to write any buffered data.

Figure 33 shows the request block for the `FPFlushFork` command.

Figure 33 Request block for the `FPFlushFork` command



FPGetACL

Gets the access control list for a file or directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
unsigned short Bitmap
long MaxReplySize
byte Pathtype
string Pathname
```

Parameters

CommandCode

`kFPGetACL` (73).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Bits that specify the values that are to be obtained. Specify `kFileSec_UUID` to get the UUID of the specified file or directory. Specify `kFileSec_GRPUUID` to get the Group UUID of the specified file or directory, or specify `kFileSec_ACL` to get the ACL of the specified file or directory. For declarations of these constants, see Access Control List Bitmap (page 169).

MaxReplySize

Reserved. Set this parameter to zero.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname of the file or directory for which the access control list (ACL) is to be obtained. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 24 for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 26 (page 74) for the format of the reply block.

Discussion

Depending on the bits set in the `Bitmap` parameter, this command gets the UUID, Group UUID, or ACL for the specified file or directory. If the `kFileSec_UUID` bit is set, the file or directory's UUID appears first in the reply packet. If the `kFileSec_GRPUUID` bit is set, the file or directory's Group UUID appears next in the reply packet. If the `kFileSec_ACL` bit is set, the file or directory's ACL appears next in the packet.

Support for this command, as well as `FPAccess` (page 11) and `FPSetACL` (page 140) is required in order to support access control lists (ACLs). Support for UTF-8 and UUIDs is also required in order to support ACLs.

Table 24 lists the result codes for the `FPGetACL` command.

Table 24 Result codes for the `FPGetACL` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access rights required to get the ACL for the specified file or directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

Table 25 describes the reply block for the `FPGetACL` command.

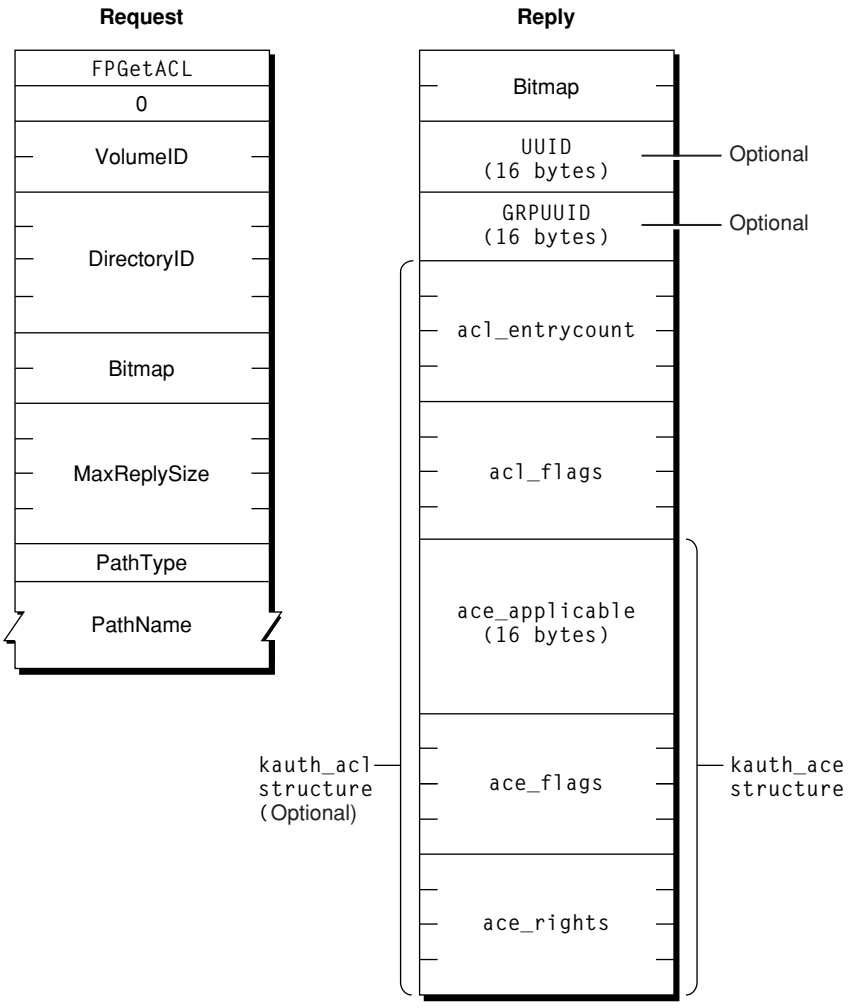
Table 25 Reply block for the `FPGetACL` command

Name and size	Data
`Bitmap` (short)	Copy of the bitmap sent to the server.

Name and size	Data
UUID (16 bytes)	File or directory's UUID, if the kFileSec_UUID bit was set.
GRPUUID (16 bytes)	File or directory's Group UUID, if the kFileSec_GRPUUID bit was set.
ACL	File or directory's ACL in a kauth_acl structure, if the kFileSec_ACL bit was set. For details, see Access Control List Structure (page 161).

Figure 34 shows the request and reply blocks for the FPGetACL command.

Figure 34 Request and reply blocks for the FPGetACL command



Version Notes
Introduced in AFP 3.2.

FPGetAPPL

Retrieves an APPL mapping from a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long FileCreator
short APPLIndex
```

Parameters

CommandCode

kFPGetAPPL (55).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

FileCreator

File creator of the application corresponding to the APPL mapping to be retrieved.

APPLIndex

Index of the APPL mapping to be retrieved.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or Desktop database reference is unknown, kFPItemNotFound if no entries in the Desktop database match the input parameters, kFPBitmapErr if an attempt was made to retrieve a parameter that cannot be obtained with this command, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 26 (page 74) for the format of the reply block.

Discussion

For each file creator, the Desktop database contains a list of APPL mappings. Each APPL mapping contains the Parent Directory ID and CNode name of an application associated with the file creator, as well as an APPL Tag that can be used to distinguish among the APPL mappings (the APPL Tag is uninterpreted by the Desktop database).

Information about the application file associated with each APPL mapping can be obtained by sending successive FPGetAPPL commands with Index varying from one to the total number of APPL mappings stored in the Desktop database for that file creator. If Index is more than the number of APPL mappings in the Desktop database for FileCreator, a kFPItemNotFound result code is returned. An Index of zero returns the first APPL mapping, if one exists in the Desktop database.

The server retrieves the specified parameters for the application file and packs the, in bitmap order, in the reply block.

The user must have search access to all ancestors except the parent directory and read access to the parent directory of the application for which information will be returned.

The user must have previously called FPOpenDT (page 120) for the corresponding volume.

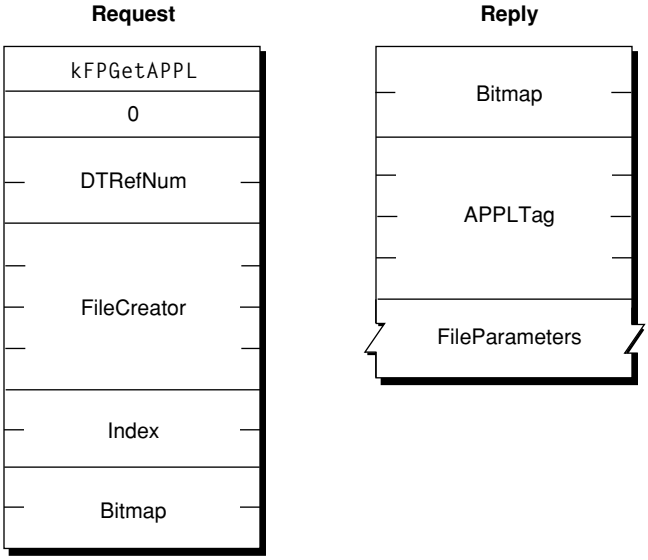
Table 26 describes the reply block for the FPGetAPPL command.

Table 26 Reply block for the `FPGetAPPL` command

Name and size	Data
`Bitmap` (short)	Bitmap describing the parameters of the application file to return. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command. For bit definitions for the File bitmap, see Table 1-7 in File Bitmap (page 164).
`APPLTag` (long)	Tag information associated with the APPL mapping.
`FileParameters`	Requested file parameters.

Figure 35 shows the request and reply blocks for the `FPGetAPPL` command.

Figure 35 Request and reply blocks for the `FPGetAPPL` command



FPGetAuthMethods

Gets the UAMs that an Open Directory domain supports.

```
byte CommandCode
byte Pad
byte Flags
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPGetAuthMethods` (62).

Pad

Pad byte.

Flags

Flags providing additional information. (No flags are currently defined.)

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname of the Open Directory domain for which UAMs are to be obtained. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred, `kFPObjectNotFound` if the specified Open Directory domain could not be found, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 27 (page 75) for the format of the reply block.

Discussion

This command gets the UAMs for the specified Open Directory domain.

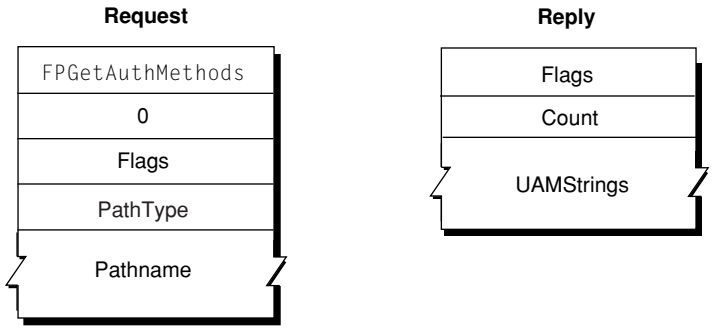
Table 27 describes the reply block for the `FPGetAuthMethods` command.

Table 27 Reply block for the `FPGetAuthMethods` command

`Flags` (byte)	Copy of the Flags input parameter.
`Count` (byte)	Number of UAMs that follow.
`UAMStrings` (packed Pascal strings)	Packed Pascal strings containing the names of the available UAMs

Figure 36 shows the request and reply blocks for the `FPGetAuthMethods` command.

Figure 36 Request and reply blocks for the `FPGetAuthMethods` command



FPGetComment

Gets the comment associated with a file or directory from the volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPGetComment` (58).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

DirectoryID

Directory ID.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 28 for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. The reply block consists of a string, called `Comment`, containing the comment text.

Discussion

The comment for the specified file or directory, if it is found in the volume's Desktop database, is returned in the reply block.

If the comment is associated with a directory, the user must have search access to all ancestors, including the parent directory. If the comment is associated with a file, the user must have search access to all ancestors except the parent directory and read access to the parent directory.

The user must have previously called `FPOpenDT` (page 120) for the corresponding volume. In addition, the file or directory must exist before this command is sent.

Table 28 lists the result codes for the `FPGetComment` command.

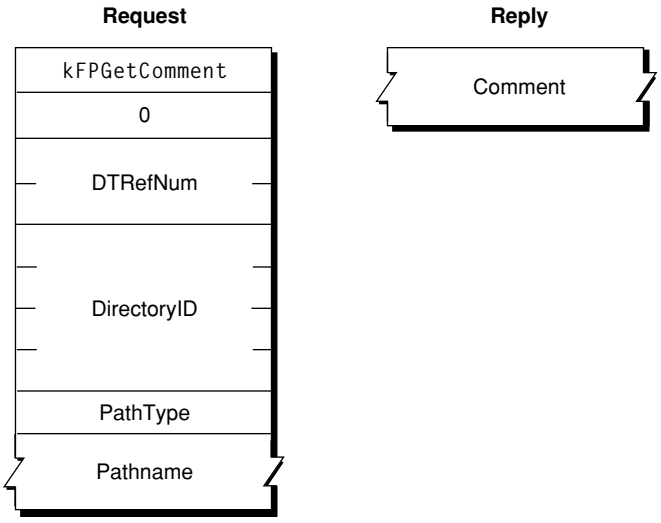
Table 28 Result codes for the `FPGetComment` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPItemNotFound`	No comment was found in the Desktop database.
`kFPParamErr`	Session reference number or Desktop database reference number is unknown.
`kFPObjectNotFound`	Input parameters do point to an existing file or directory.

Result code	Explanation
kFPMiscErr	Non-AFP error occurred.

Figure 37 shows the request and reply blocks for the FPGetComment command.

Figure 37 Request and reply blocks for the FPGetComment command



FPGetExtAttr

Gets the value of an extended attribute.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID unsigned short Bitmap
long long Offset
long long ReqCount
long MaxReplySize
byte PathType
string Pathname
byte Pad
unsigned short NameLength
string Name
```

Parameters

CommandCode

kFPGetExtAttr (69).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID
Directory identifier.

Bitmap
Bitmap specifying the desired behavior when getting the value of an extended attribute. For this command, only `kAttrDontFollow` is valid. For details, see `Extended Attributes Bitmap` (page 164) for details.

Offset
Always zero; reserved for future use.

ReqCount
Always –1; reserved for future use.

MaxReplySize
Size in bytes of the reply that your application can handle; set to zero to get the size of the reply without actually getting the attributes.

PathType
Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname
Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Pad
Optional pad byte if needed to pad to an even boundary.

NameLength
Length in bytes of the extended attribute name that follows.

Name
UTF-8–encoded name of the extended attribute whose value is to be obtained.

Result
`kFPNoErr` if no error occurred. See Table 28 for other possible result codes.

ReplyBlock
If the result code is `kFPNoErr`, the server returns a reply block. The reply block consists of a bitmap and the value of the extended attribute that was requested. See Table 30 (page 79) for the format of the reply block.

Discussion
If the result code is `kFPNoErr`, this command returns in the reply block the value of the extended attribute that was requested.

Support for this command, as well as `FPListExtAttrs` (page 103), `FPRemoveExtAttr` (page 134) and `FPSetExtAttr` (page 145) is required in order to support extended attributes. UTF-8 support is also required in order to support extended attributes.

Table 29 lists the possible result codes for the `FPGetExtAttr` command.

Table 29 Result codes for the `FPGetExtAttr` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to get the file of an extended attribute for the specified file or directory.
`kFPBitmapErr`	Bitmap is null or specifies a value that is invalid for this command.

Result code	Explanation
kFPMiscErr	Non-AFP error occurred.
kFPObjectNotFound	Input parameters do not point to an existing file or directory.
kFPParamErr	A parameter is invalid.

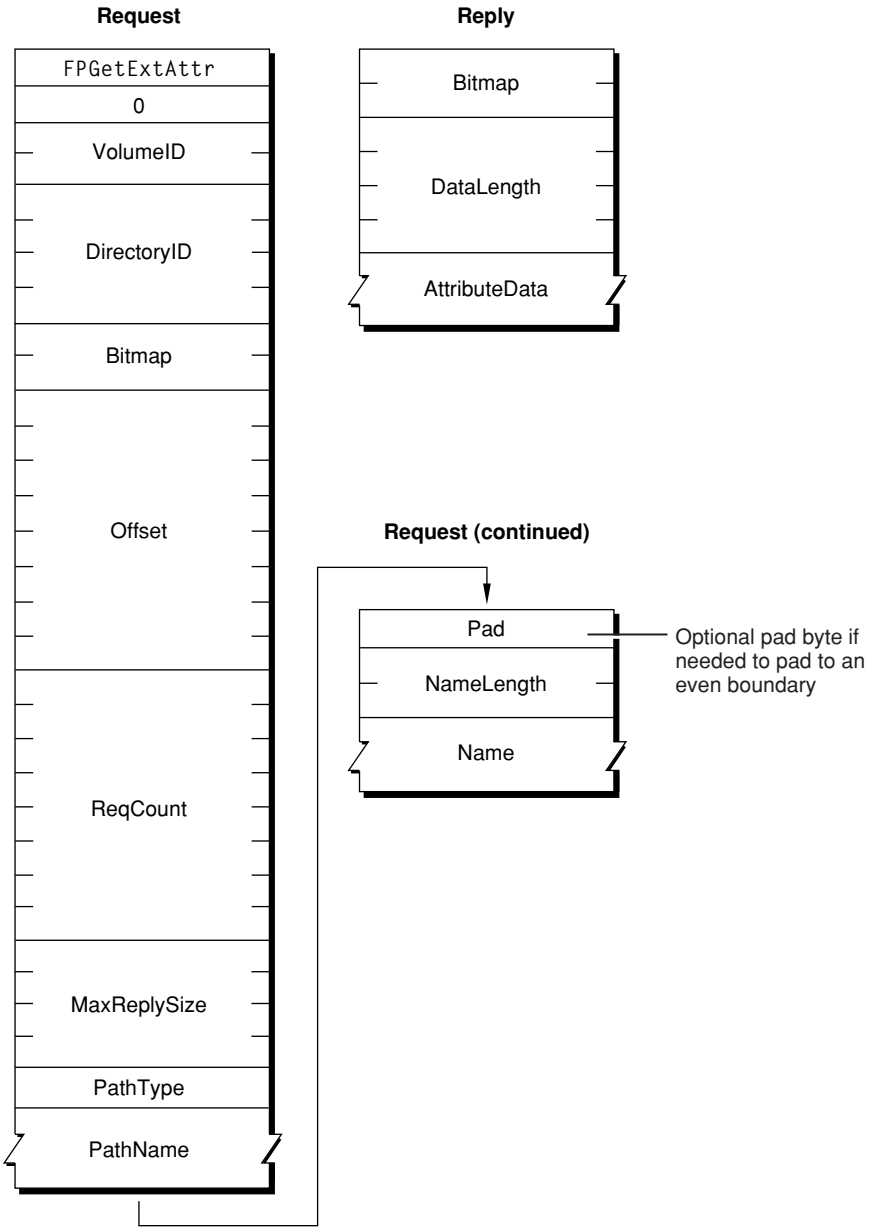
Table 30 describes the reply block for the FPGetExtAttr command.

Table 30 Reply block for the FPGetExtAttr command

Name and size	Data
Bitmap (short)	Copy of the input parameter.
DataLength (long)	Length in bytes of the extended attribute data that follows.
ExtendedAttribute Data (string)	Extended attribute data

Figure 37 shows the request and reply blocks for the FPGetExtAttr command.

Figure 38 Request and reply blocks for the `FPGetExtAttr` command



Version Notes
Introduced in AFP 3.2.

FPGetFileDirParms

Gets the parameters for a file or a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short FileBitmap
short DirectoryBitmap
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPGetFileDirParms` (34).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Directory ID.

FileBitmap

Bitmap describing the parameters to return for a file. Set the bit that corresponds to each desired parameter. For the bit definitions of this bitmap, see File Bitmap (page 164).

DirectoryBitmap

Bitmap describing the parameters to return for a directory. Set the bit that corresponds to each desired parameter. For the bit definitions of this bitmap, see Directory Bitmap (page 162).

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 31 (page 83) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 32 (page 83) for the format of the reply block.

Discussion

The server packs the requested parameters in the reply block in the order specified by the appropriate bitmap. The `FileDir` bit indicates whether the parameters are for a file or a directory. A copy of the input bitmaps is inserted before the parameters.

Variable-length parameters, such as Long Name and Short Name, are kept at the end of the block. To do this, the server represents variable-length parameters in the bitmap order as fixed-length offsets (integers). Each offset is measured from the start of the parameters (not from the start of the bitmap) to the start of the variable-length parameter. The actual variable-length parameters are then packed after all fixed-length parameters.

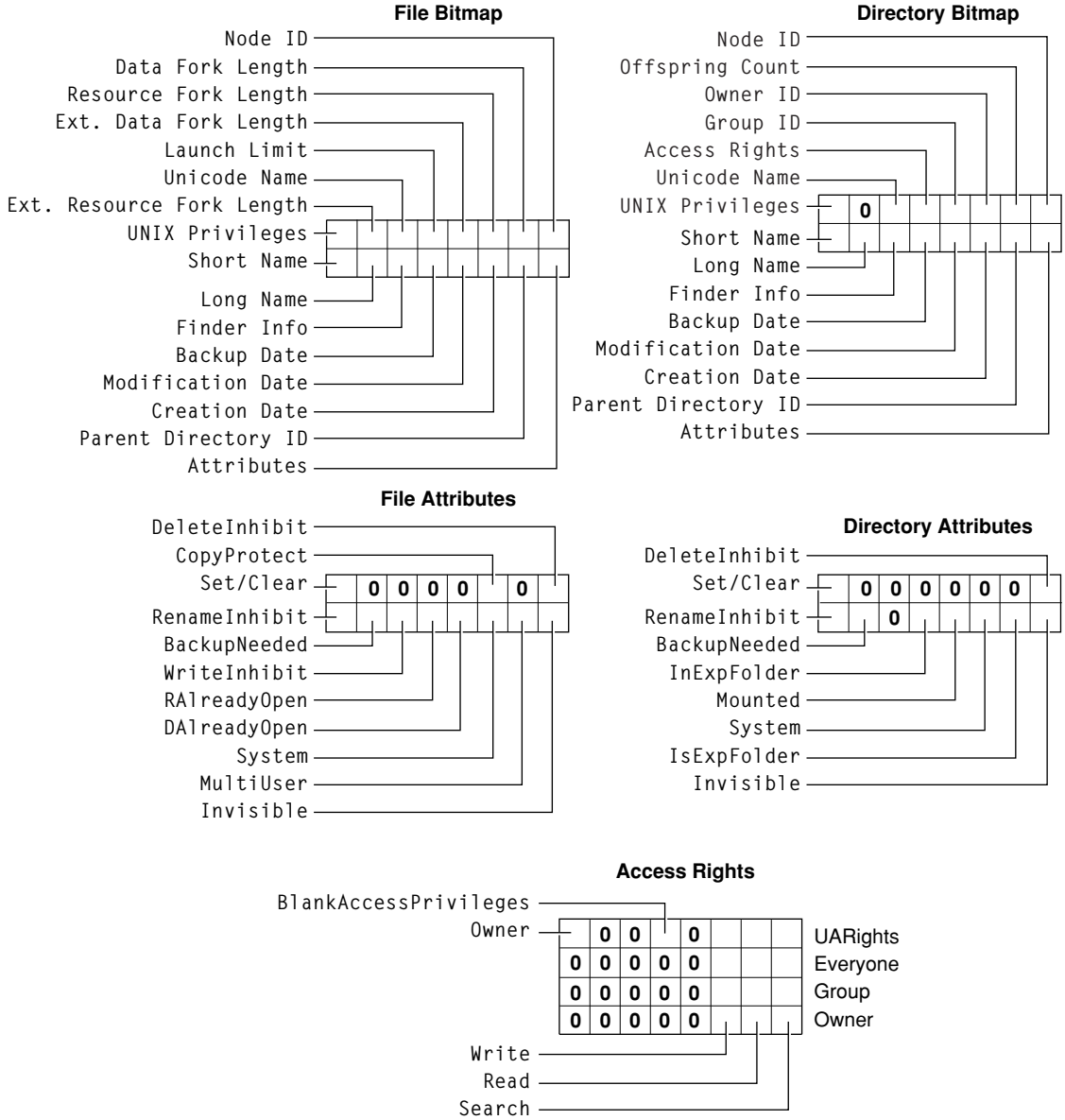
If the CNode exists and both bitmaps are null, no error is returned; `FileBitmap`, `DirectoryBitmap`, and the byte containing the `FileDir` bit are returned with no other parameters.

If a directory's access rights are requested, the server returns the Access Rights parameter (a four-byte quantity) containing the read, write, and search access privileges corresponding to owner, group, and everyone as well as the User Access Rights Summary byte, which indicates the privileges the current user of the AFP client has to this directory. For bit definitions of the Access Rights parameter, see Access Rights Bitmap (page 162).

If the Offspring Count bit of the `DirectoryBitmap` parameter is set, the server will adjust the Offspring Count to reflect the access rights the user has to that directory. For example, if a particular directory contains three file and two directory offspring, the server will return its Offspring Count as two if the user has only search access to the directory, three if the user has only read access to the directory, or five if the user has both search and read access to the directory.

Figure 39 (page 82) shows the File and Directory bitmaps, the File and Directory Attributes parameters, and the Access Rights for directories.

Figure 39 Bitmaps, Attributes, and Access Rights returned by `FPGetFileDirParms`



The user must have search access to all ancestors except this CNode's parent directory. For directories, the user also needs search access to the parent directory. For files, the user needs read access to the parent directory.

Most of the attributes requested by this command are stored in corresponding flags within the CNode's Finder Info record.

Table 31 lists the result codes for the `FPGetFileDirParms` command.

Table 31 Result codes for the `FPGetFileDirParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to retrieve a parameter that cannot be obtained with this command.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is invalid.

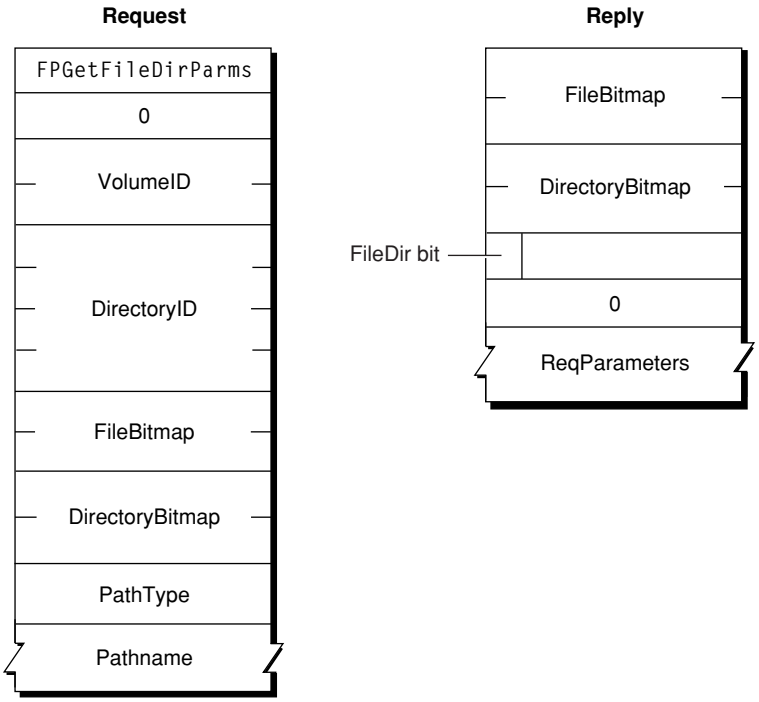
Table 32 describes the reply block for the `FPGetFileDirParms` command.

Table 32 Reply block for the `FPGetFileDirParms` command

Name and size	Data
`FileBitmap` (short)	Copy of the input parameter.
`DirectoryBitmap` (short)	Copy of the input parameter.
`FileDir` (bit)	Bit indicating whether the CNode is a file or a directory:0 = file 1 = directory
`ReqParameters`	Requested parameters.

Figure 40 shows the request and reply blocks for the `FPGetFileDirParms` command.

Figure 40 Request and reply blocks for the `FPGetFileDirParms` command



FPGetForkParms

Gets the parameters for a fork.

```
byte CommandCode
byte Pad
short OForkRefNum
short Bitmap
```

Parameters

CommandCode

`kFPGetForkParms` (14).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Bitmap

Bitmap describing the parameters to be returned. Set the bits that correspond to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command. For bit definitions for this bitmap, see File Bitmap (page 164).

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or open fork reference number is invalid, `kFPBitmapErr` if an attempt was made to retrieve a parameter that cannot be obtained with this command; bitmap is null, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 33 (page 85) for the format of the reply block.

Discussion

The server packs the parameters in bitmap order in the reply block.

Variable-length parameters, such as Long Name and Short Name, are kept at the end of the block. To do this, the server represents variable-length parameters in the bitmap order as fixed-length offsets (integers). Each offset is measured from the start of the parameters (not from the start of the bitmap) to the start of the variable-length parameter. The actual variable-length fields are then packed after all fixed-length parameters.

This command retrieves the length the fork indicated by `OForkRefNum`; a `kFPBitmapErr` result code is returned if an attempt is made to retrieve the length of the file's other fork.

The user must have previously called `FPOpenFork` (page 121) for this volume.

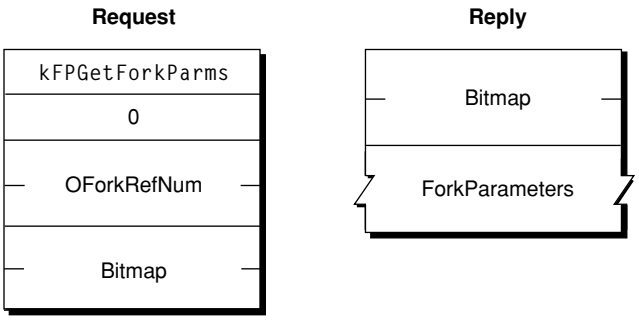
Table 33 describes the reply block for the `FPGetForkParms` command.

Table 33 Reply block for the `FPGetForkParms` command

Name and size	Data
`Bitmap` (short)	Copy of the input parameter.
`FileParameters`	Requested fork parameters.

Figure 41 shows the request and reply blocks for the `FPGetForkParms` command.

Figure 41 Request and reply blocks for the `FPGetForkParms` command



FPGetIcon

Gets an icon from the Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long FileCreator
long FileType
byte IconType
byte Pad
short Length
```

Parameters

CommandCode

kFPGetIcon (51).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

FileCreator

File creator of the file with which the icon is associated.

FileType

File type of the file with which the icon is associated.

IconType

Preferred icon type.

Pad

Pad byte.

Length

Number of bytes the caller expects the icon bitmap to require in the reply block.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or Desktop database reference number is unknown, kFPItemNotFound if no icon corresponding to the input parameters was found in the Desktop database, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. The reply block consists of a short containing the requested icon bitmap.

Discussion

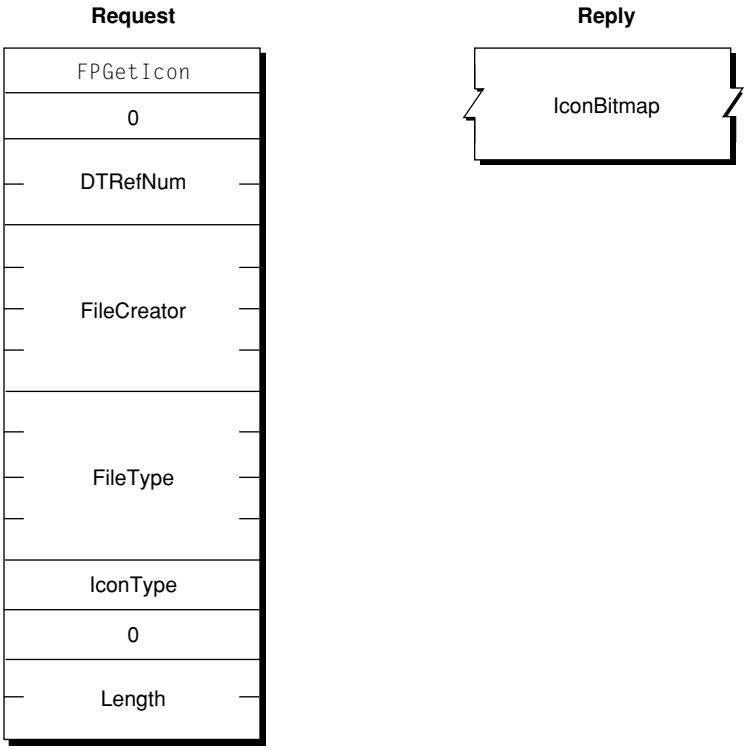
The server retrieves an icon bitmap from the Desktop database as specified by the FileCreator, FileType, and IconType parameters.

An input Length value of zero is acceptable to test for the presence or absence of a particular icon. If Length is less than the actual size of the icon bitmap, only Length bytes are returned.

The user must have previously called FPOpenDT (page 120) for the corresponding volume.

Figure 42 shows the request and reply blocks for the FPGetIcon command.

Figure 42 Request and reply blocks for the `FPGetIcon` command



FPGetIconInfo

Gets icon information from the Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long FileCreator
short IconIndex
```

Parameters

CommandCode

`kFPGetIconInfo` (51).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

FileCreator

File creator of the file with which the icon is associated.

IconIndex

Index of the requested icon.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or Desktop database reference number is unknown, kFPItemNotFound if no icon corresponding to the input parameters was found in the Desktop database, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 34 (page 88) for the format of the reply block.

Discussion

The server retrieves a information about an icon in the volume's Desktop database as specified by the icon's file creator and icon index.

For each file creator, the Desktop database contains a list of icons. Information about each icon can be obtained by sending successive FPGetIconInfo commands with IconIndex varying from one to the total number of icons stored in the Desktop database for that file creator. If IconIndex is more than the number of icons in the Desktop database for the specified file creator, a result code of kFPItemNotFound is returned.

The user must have previously called FPOpenDT (page 120)FPOpenDT (page 120) for the corresponding volume.

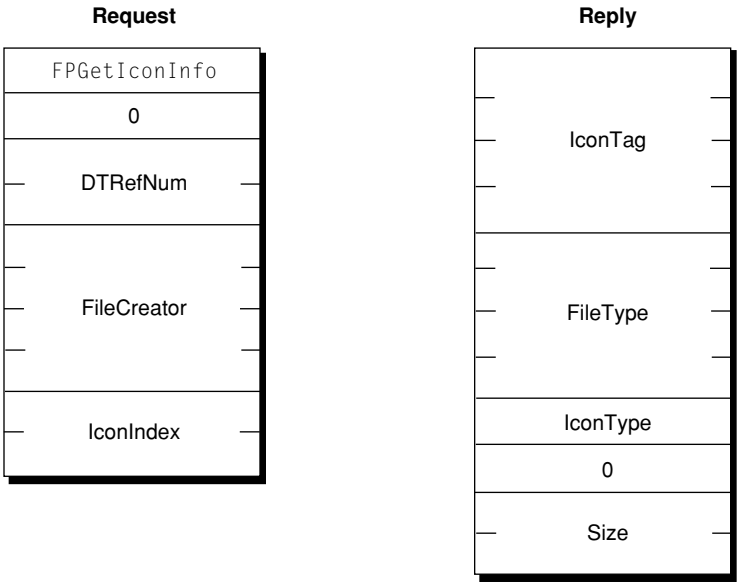
Table 34 describes the reply block for the FPGetIconInfo command.

Table 34 Reply block for the FPGetIconInfo command

Name and size	Data
IconTag (long)	Tag information associated with the requested icon.
FileType (long)	File type of the requested icon.
IconType (byte)	Type of the requested icon.
Pad (byte)	Pad byte.
Size (short)	Size of the icon bitmap.

Figure 43 shows the request and reply blocks for the FPGetIconInfo command.

Figure 43 Request and reply blocks for the `FPGetIconInfo` command



FPGetSessionToken

Gets a session token.

```
byte CommandCode
byte Pad
short Type
long IDLength
long timeStamp (optional)
ID
```

Parameters

CommandCode

`kFPGetSessionToken` (64).

Pad

Pad byte.

Type

The value of this parameter is `kLoginWithoutID` (0) if the client supports an earlier version of AFP that does not send an `IDLength` and an `ID` parameter. It is `kLoginWithTimeAndID` (3) if the client is sending an `IDLength`, an `ID`, and a `Timestamp` parameter and the client wants its old session to be discarded. It is `kReconnWithTimeAndID` (4) if the client has just finished a successful reconnect, is sending an `IDLength`, an `ID`, and a `Timestamp` parameter, and wants the session to be updated with the `ID` parameter. It is `kGetKerberosSessionKey` (8) if the client is logging in using Kerberos v5. See FPGetSessionToken Types (page 171) for the enumeration that defines the constants for this parameter.

IDLength

Length of the `ID` parameter.

timeStamp

Optional time stamp specified only if the value of `ID` is `kLoginWithTimeAndID` or `kReconnWithTimeAndID`.

ID

A client-defined value that uniquely identifies this session.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number is unknown, `kFPCallNotSupported` if the server does not support this command, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 35 (page 91) for the format of the reply block.

Discussion

This command helps an AFP client manage a disconnect that occurs and there are open files or locked resources on the remote server. The remote server will save the current state of the session (including open files) and wait until its reconnect timeout expires before closing the files and discarding the saved session. In the case of an AFP client that fails to wake up properly from sleep with a mounted AFP server, the session will be saved on the remote server until the sleep timeout expires.

Under these circumstances, prior to AFP 3.1, when an AFP client logged back into the server from the same system, attempts to access the files that were open at the time of the crash would fail with a "file already open" or "resources already locked" result. The AFP client would have to wait for the reconnect or sleep timeout to expire, or a server administrator would have to manually disconnect the old session.

With AFP 3.1, the AFP client can set the `Type` parameter to `kLoginWithID`, set the `ID` parameter to a unique client-defined value, and send this command. The server will associate the value of `ID` with the session. Later, if the local system crashes and the AFP client logs in again, the AFP client can set the `Type` parameter to `kLoginWithID`, set the `ID` parameter to the ID that it previously sent to the remote server, and send this command again. The remote server will look for a session that matches the value of ID. If a match is found, it will close any files associated with the session that are open, free any locked resources, and disconnect the matching session. Note that in the current version, the unique ID is associated with a particular computer, so after the system crashes, the AFP client must log in from the same computer using the same information that was used to log in originally.

With AFP 3.1, the AFP client can also set the `Type` parameter to `kLoginWithTimeAndID`. In this case, the client must include a four-byte time stamp after the `IDLength` field, and the server saves the time stamp as well as the value of `ID` for each session. When the server receives a `FPGetSessionToken` command having a `TYPE` parameter whose value is `kLoginWithTimeAndID`, the server searches all of its session queues. If the server finds a session that matches the value of `ID`, it also checks the time stamp. If the time stamp matches, the client has *not* restarted so the session is not discarded. The session is only discarded if the saved time stamp does not match.

This command returns a session token that, with AFP 3.0 and later, is used to reconnect. If the local system is disconnected and the AFP client logs in again using the same log in information as before, the AFP client can call `FPDisconnectOldSession` (page 52), passing the session token obtained by calling `FPGetSessionToken`, to tell the server to transfer the old session to the new session.

Note that sending the `FPGetSrvrMsg` command does not initiate a reconnect.

For security purposes, the server always fails reconnections for users who log in as Guest.

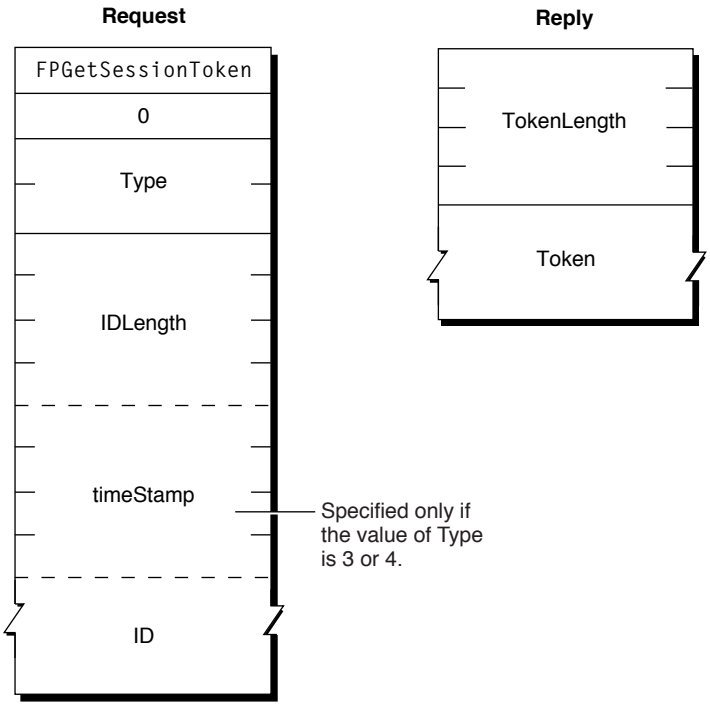
Table 35 describes the reply block for the `FPGetSessionToken` command.

Table 35 Reply block for the `FPGetSessionToken` command

Name and size	Data
`TokenLength` (long)	Length of the token that follows.
`Token` (variable length)	Token that can be passed to `FPDisconnectOldSession` if the session is inadvertently disconnected and then re-established. If the client supports the Reconnect UAM, the token needs to be refreshed periodically, and is also sent to the server by the `FPLoginExt` (page 109) command to reconnect if the session is disconnected.

Figure 44 shows the request and reply blocks for the `FPGetSessionToken` command.

Figure 44 Request and reply blocks for the `FPGetSessionToken` command



FPGetSrvrInfo

Gets information about a server.

```
byte CommandCode
byte Pad
```

Parameters

CommandCode

`kFPGetSrvrInfo` (15).

Pad

Pad byte.

Result

`kFPNoErr` if no error occurred, `kFPNoServer` if the server is not responding, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 37 (page 94) for the format of the reply block.

Discussion

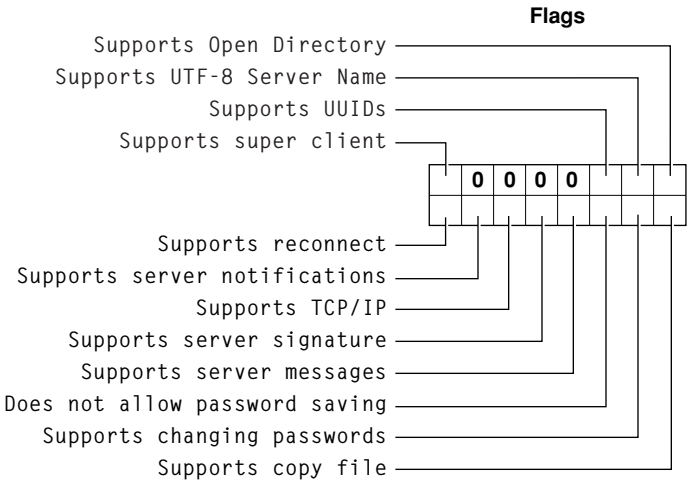
The reply block begins with the offset to the `MachineType` parameter, followed by the offset to the `AFPVersions` parameter, the offset to the `UAMs` parameter, and the offset to the `VolumeIconAndMask` parameter. The offsets are followed by the `Flags` parameter, the `ServerName` parameter padded to an even boundary, the offset to the `ServerSignature` parameter, and the offset to the `NetworkAddresses` parameter.

The server packs the information in the reply block in any order, so no assumption should be made about the order of the information. Instead AFP clients should access the information only through the offsets. The exception is the `ServerName` parameter, which is always after the `Flags` parameter.

Providing offsets to the `VolumeIconAndMask`, `ServerSignature`, `NetworkAddresses`, and `DirectoryNames` parameters is required, but providing the parameters themselves is optional. If not provided, the value of each parameter is zero.

The `Flags` parameter indicates the server's support for certain features. If bit 9 in the `Flags` parameter is set, the reply block contains a `UTF-8ServerNames` offset to the server's name in UTF-8 format. Figure 45 shows how bits are used in the `Flags` parameter.

Figure 45 Bit usage in the ServerFlags parameter



The `AFPVersionsCount` and `UAMCount` parameters are each one byte containing the number of AFP and UAM version strings that follow, with the strings packed back-to-back without padding. For the AFP versions supported by this version of AFP, see `AFP Version Strings` (page 170). For the UAMs supported by this version of AFP, see `AFP UAM Strings` (page 171).

The optional `ServerSignature` parameter contains a unique identifier for the server. An AFP client should use the server signature to ensure that it does not log on to the same server multiple times. Preventing multiple log ins is important when the server is configured for multihoming.

The `NetworkAddresses` parameter contains the addresses that the client can use to connect to the server. Each address is stored as an AFP Network Address. The format of an AFP Network Address is shown in Figure 46.

Figure 46 AFP Network Address format

Each AFP Network Address consists of a length byte containing the total length in bytes of the Network Address, followed by a tag byte identifying the type of address the Address field contains, followed by up to 254 bytes of data. Table 36 lists the possible values of the `Length` and `Tag` fields and describes the type of address stored in the `Address` field.

Table 36 AFP Network Address fields

Length	Tag	Address
	0x00	Reserved
0x6	0x01	Four-byte IP address
0x8	0x02	Four-byte IP address followed by a two-byte port number
0x6	0x03	DDP address (two bytes for the network number, one byte for the node number, and one byte for the socket number)
Variable	0x04	DNS name
0x8	0x05	IP address (four bytes) with port number (2 bytes). If this tag is present and the client is so configured, the client attempts to build a Secure Shell (SSH) tunnel between itself and the server and try to connect through the it.
0x12	0x6	IPv6 address (16 bytes)
0x14	0x07	IPv6 address (16 bytes) following by a two-byte port number

The network address format provides the available network addresses to the AFP client. AFP clients should ignore any tags that it does not recognize.

`FPGetSrvrInfo` can be called without first establishing a session with the server.

Table 37 describes the reply block for the `FPGetSrvrInfo command`.

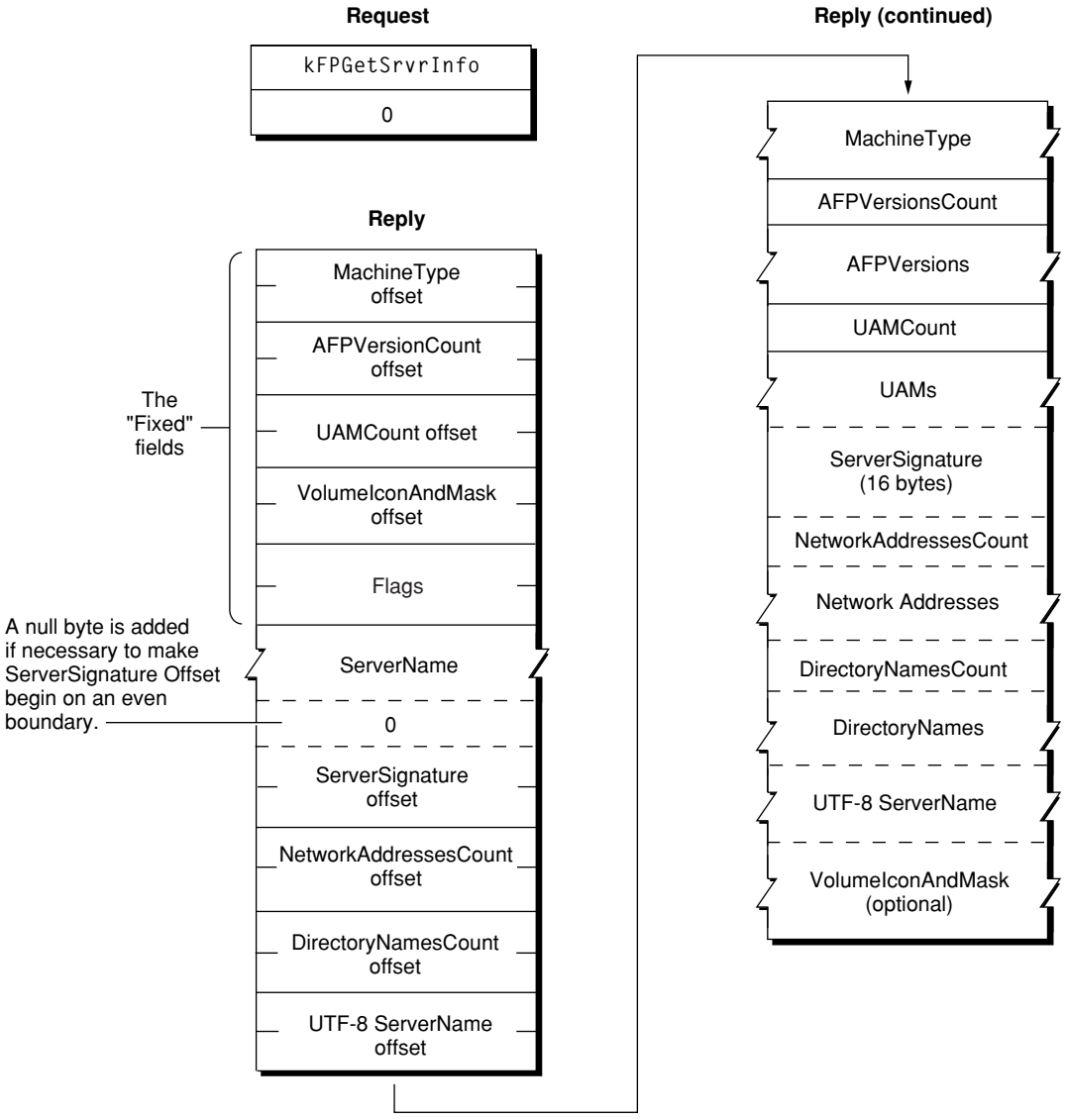
Table 37 Reply block for the `FPGetSrvrInfo` command

Name and size	Data
`MachineType` offset	Offset to the location in the reply block containing the server's machine type.
`AFPVersionCount` offset	Offset to the location in the reply block containing the number of AFP versions the server supports.
`UAMCount` offset	Offset to the location in the reply block containing the number of UAMs the server supports.
`VolumeIconAndMask` offset	Offset to the location in the reply block containing volume icon and mask data.
`Flags` (short)	Flags describing the server's capabilities. For bit definitions, see the section Server Flags Bitmap (page 167).
`ServerName` (string)	String containing the server's name.
`ServerSignature` offset	Offset to the location in the reply block containing the server's signature.
`NetworkAddressesCount` offset	Offset to the location in the reply block containing the number of AFP Network Addresses.
`DirectoryNamesCount` offset	Offset to the location in the reply block containing the number of Directory Names.
`UTF-8ServerName`	Offset to the location in the reply block containing the server's name in UTF-8 characters.
`MachineType` (string)	A string containing a description of the server's hardware, operating system, or both.
`AFPVersionsCount`	Number of AFP version strings that follow.
`AFPVersions` (packed string)	Each AFP version that the server supports in packed format. For each supported version, there is one byte stating the number of bytes in the version string that follows.
`UAMsCount`	Number of UAM strings that follow.
`UAMs` (packed string)	Each UAM that the server supports in packed format. For each supported UAM, there is one byte stating the number of bytes in the UAM name string that follows.
`ServerSignature` (16 bytes)	Sixteen-byte number that uniquely identifies the server, or zero if not supported. For AFP servers, supporting server signatures is optional, but AFP servers must provide a `ServerSignature` offset. An AFP server indicates that it supports server signatures by setting the `kSupportsSrvrSig` bit in the Flags parameter.

Name and size	Data
`NetworkAddresses` (AFP Network Address)	Server's network addresses, or zero if not supported. (The AFP Network Address format is described later in this section.) For AFP servers, providing a `NetworkAddresses` parameter is optional, but AFP servers must provide a `NetworkAddresses` offset. An AFP server indicates that it supports Network Addresses by setting the `kSupportsTCP` bit in the `Flags` parameter.
`DirectoryNames` (string)	String containing the names of directories that Open Directory has made available for sharing, or zero if not supported. For AFP servers, supporting Open Directory is optional, but AFP servers must provide a `DirectoryNames` offset. An AFP server indicates that it supports Open Directory by setting the `kSupportsDirServices` bit in the Flags parameter. If the server supports the Kerberos UAM, it places its principal name in this string.
`UTF-8ServerName` (AFPName)	AFPName containing the UTF-8–encoded name of the server.
`VolumeIconAndMask` (256 bytes)	128 bytes of icon data and 128 bytes of mask data.

Figure 47 shows the request and reply blocks for the `FPGetSrvrInfo` command.

Figure 47 Request and reply blocks for the `FPGetSrvrInfo` command



FPGetSrvrMsg

Gets a message from a server.

```
byte CommandCode
byte Pad
short MessageType
short MessageBitmap
```

Parameters

CommandCode

`kFPGetSrvrMsg` (38).

Pad

Pad byte.

MessageType

Type of message, were 0 indicates log in and 1 indicates server. (Set `MessageType` to 1 when the Server Message bit in the attention code is set.)

MessageBitmap

Bitmap providing additional information. The client sets bit 0 of this bitmap to indicate it is requesting a message. Starting with AFP 3.0, the client can set bit 1 of this bitmap to indicate that it supports UTF-8 messages.

Result

`kFPNoErr` if no error occurred, `kFPCallNotSupported` if the server does not support this command, `kFPBitmapErr` if unrecognized bits are set in `MessageBitmap`, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 38 (page 98) for the format of the reply block.

Discussion

An AFP client uses the `FPGetSrvrMsg` command to get messages from the server. Usually, the server sends an attention code to the client when server messages are available, and the client responds by calling `FPGetSrvrMsg`. However, the client can call `FPGetSrvrMsg` at any time. If no message is available when the client calls `FPGetSrvrMsg`, the server returns a zero-length string.

There are two message types: log in and server. The log in message type allows the server to send a message to a client at log in time. The client can query the server for a log in message at log in time, or whenever it is convenient to do so. If there is no login message, `FPGetSrvrMsg` returns a zero-length string.

The server message type allows the server to send messages to the client once the client has logged on. The server notifies the client that a server message is available by sending a DSI Attention packet in which the Server Message bit in the `AFPUserBytes` field is set.

There are two server message types:

- Shutdown. The server can send a shutdown message to explain why the server is shutting down, how long it will be down, and so on. In addition to setting the Server Message bit in the `AFPUserBytes` field of the DSI Attention packet, the server sets the Shutdown bit to indicate that a shutdown message is available.
- User. The server can send a message to a specific user. The client is made aware that a user message is available when the server sends an DSI Attention packet in which the Server Message bit in the `AFPUserBytes` field is set and the Shutdown bit is not set.

The usual size of any of these messages is 200 bytes including the length byte (a Str199). AFP 3.*x* clients can request that the server send longer attention messages by setting the attention quantum size in the `Option` field of the `DSOpenSession` command (described in the document *Apple Filing Protocol Client*).

The user must be logged on to the server to receive server message notifications. Otherwise, no special access privileges are necessary to use this command.

Note that in the case of a disconnected session, sending this command does not initiate a reconnect.

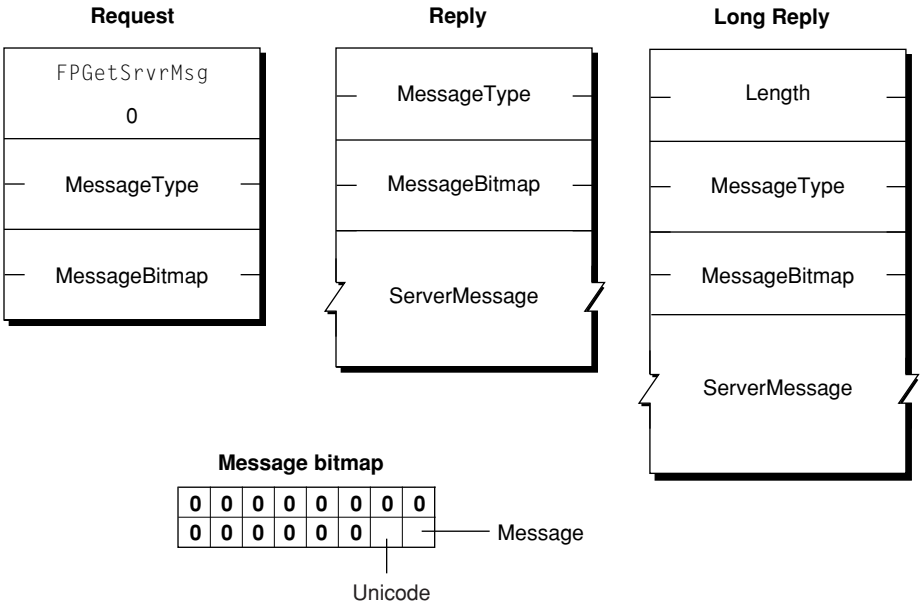
Table 38 describes the reply block for the `FPGetSrvrMsg` command.

Table 38 Reply block for the `FPGetSrvrMsg` command

Name and size	Data
`MessageType` (short)	Copy of the input parameter.
`MessageBitmap` (short)	Bitmap describing the message. Bit 0 is set if `ServerMessage` contains a message. Starting with AFP 3.0, bit 1 is set to indicate that the message is UTF-8 encoded.
`ServerMessage` (string)	Message from the server.

Figure 48 shows the request and reply blocks for the `FPGetSrvrMsg` command.

Figure 48 Request and reply blocks for the `FPGetSrvrMsg` command



FPGetSrvrParms

Gets server parameters.

```
byte CommandCode
byte Pad
```

Parameters

`CommandCode`

`kFPGetSrvrParms` (16).

`Pad`

Pad byte.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number is unknown, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 39 for the format of the reply block.

Discussion

The VolName string, the HasPassword bit, and the HasConfigInfo bit are packed without padding in the reply block.

For AFP 2.x, this command returns volume names in ANSI format with a maximum length of 27 bytes.

For AFP 3.x, this command returns volume names in UTF-8 format with a one byte length byte specifying any length up to 255. Note that the Finder limits setting volume names to no more than 27 characters.

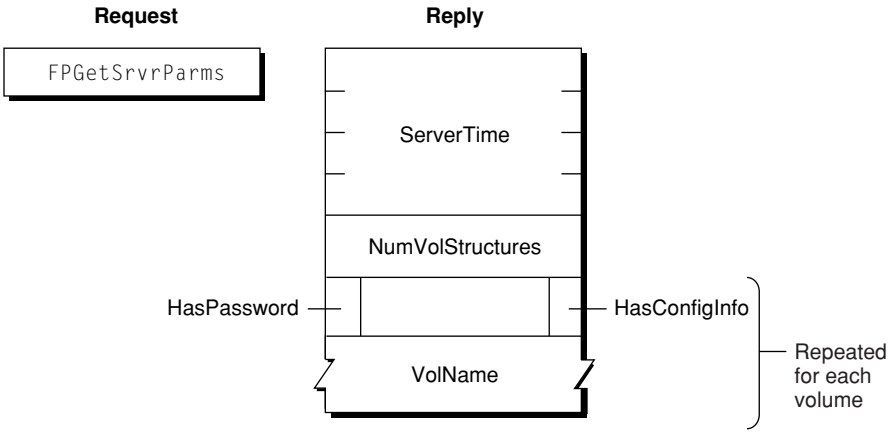
Table 39 describes the reply block for the FPGetSrvrParms command.

Table 39 Reply block for the FPGetSrvrParms command

Name and size	Data
ServerTime (long)	Current date and time on the server's clock.
MessageBitmap (short)	Number of VolStructure structures that follow.
VolStructure	An array of Volstructure structures consisting of the following fields: Flags (byte) where bit 0 (HasConfigInfo) is set if the volume has configuration information and bit 7 (HasPassword) is set if a password is set for the volume VolName (string) name of the volume

Figure 49 shows the request and reply blocks for the FPGetSrvrParms command.

Figure 49 Request and reply blocks for the FPGetSrvrParms command



FPGetUserInfo

Gets information about a user.

```
byte CommandCode
byte Pad
long UserID
short Bitmap
```

Parameters

CommandCode

kFPGetUserInfo (37).

Pad

Pad byte.

UserID

ID of user for whom information is to be retrieved. (Not valid if the ThisUser bit is set.)

Bitmap

Bitmap describing which ID to retrieve, where bit zero is set to get the user's User ID and bit 1 is set to get the user's Primary Group ID.

Result

kFPNoErr if no error occurred. See Table 40 for the other possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 41 (page 101) for the format of the reply block.

Discussion

The server retrieves the specified information for the specified user and packs them, in bitmap order, in the reply block.

This command can be used only to retrieve the User ID and the Primary Group ID of the user who is the client of this session, thus requiring that the ThisUser bit be set. The UserID parameter is intended for future use.

Table 40 lists the result codes for the FPGetUserInfo command.

Table 40 Result codes for the FPGetUserInfo command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to get information about the specified user.
kFPBitmapErr	Attempt was made to retrieve a parameter that cannot be obtained with this command.
kFPCallNotSupported	Server does not support this command.
kFPItemNotFound	Specified User ID is unknown.
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	ThisUser bit is not set.

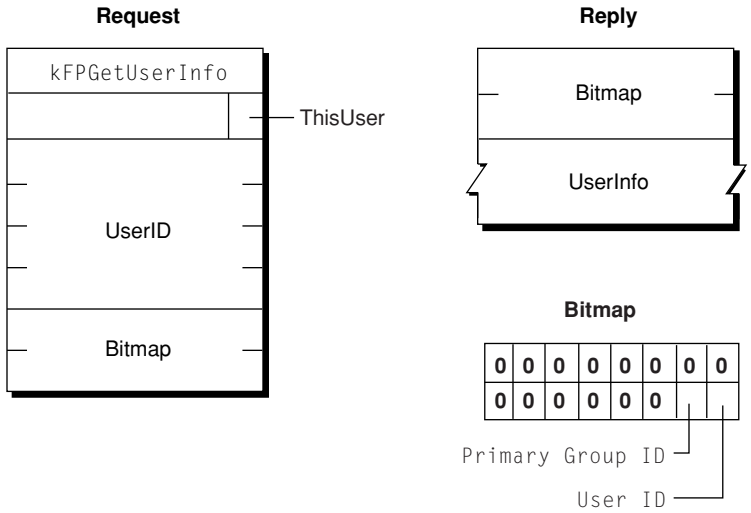
Table 41 describes the reply block for the `FPGetUserInfo` command.

Table 41 Reply block for the `FPGetUserInfo` command

Name and size	Data
`Bitmap` (short)	Copy of the input parameter.
`UserInfo`	Requested information, packed in bitmap order.

Figure 50 shows the request and reply blocks for the `FPGetUserInfo` command.

Figure 50 Request and reply blocks for the `FPGetUserInfo` command



FPGetVolParms

Gets volume parameters.

```
byte CommandCode
byte Pad
short VolumeID
short Bitmap
```

Parameters

CommandCode

`kFPGetVolParms` (17).

Pad

Pad byte.

VolumeID

Volume ID for the volume whose parameters are to be retrieved.

Bitmap

Bitmap describing the parameters that are to be returned. Set the bit of the desired parameter. This bitmap is the same as the bitmap used by the `FPOpenVol` (page 124) command. For bit definitions for this bitmap, see Volume Bitmap (page 168).

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or Volume ID is unknown, `kFPBitmapErr` if the specified bitmap has unrecognized bits set, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 42 (page 102) for the format of the reply block.

Discussion

This command retrieves parameters that describe a volume as specified by the volume's Volume ID.

The server responds to this command by returning a reply block containing a bitmap for the volume parameters and the parameters themselves. All variable-length parameters, such as Volume Name, are at the end of the block. The server represents variable-length parameters in bitmap order as fixed-length offsets (shorts). These offsets are measured from the start of the parameters (not from the start of the bitmap) to the start of the variable-length parameters. The variable-length parameters are then packed after all fixed-length parameters.

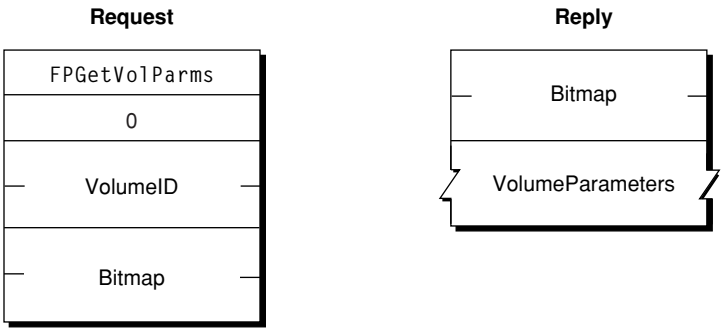
Table 42 describes the reply block for the `FPGetVolParms` command.

Table 42 Reply block for the `FPGetVolParms` command

Name and size	Data
`Bitmap` (short)	Copy of the input parameter.
`VolumeParameters`	Volume parameters packed in bitmap order.

Figure 51 shows the request and reply blocks for the `FPGetVolParms` command.

Figure 51 Request and reply blocks for the `FPGetVolParms` command



For the layout of the bitmap and the volume parameters, see the sections `Volume Attributes Bitmap` (page 167) and Volume Bitmap (page 168).

FPListExtAttrs

Gets the names of extended attributes for a file or directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID unsigned short Bitmap
short ReqCount
long StartIndex
long MaxReplySize
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPListExtAttrs` (72).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Bitmap describing the desired behavior when getting the names of extended attributes. For this command `kAttrDontFollow` is the only valid bit. For details, see `Extended Attributes Bitmap` (page 164).

ReqCount

Reserved for future use. For AFP 3.2, clients can set this parameter to any numeric value. Servers should ignore this parameter and return all extended attribute names.

StartIndex

Reserved for future use. For AFP 3.2, set `StartIndex` to zero. Servers should ignore this parameter.

MaxReplySize

Size in bytes of the reply that your application can handle, including the size of the `Bitmap` and `DataLength` parameters. Set this parameter to zero to get the size of the reply block that would be returned without actually getting the names of the extended attributes.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 43 for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See `"Reply block for the FPListExtAttrs command "` for the format of the reply block.

Discussion
If the result code is `kFPNoErr`, this command returns in the reply block the names of extended attributes for the specified file or directory.

Support for this command, as well as `FPGetExtAttr` (page 77), `FPRemoveExtAttr` (page 134) and `FPSetExtAttr` (page 145) is required in order to support extended attributes. UTF-8 support is also required in order to support extended attributes.

Table 43 lists the possible result codes for the `FPListExtAttrs` command.

Table 43 Result codes for the `FPListExtAttrs` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to list the extended attribute names for the specified file or directory.
`kFPBitmapErr`	Bitmap is null or specifies a value that is invalid for this command.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

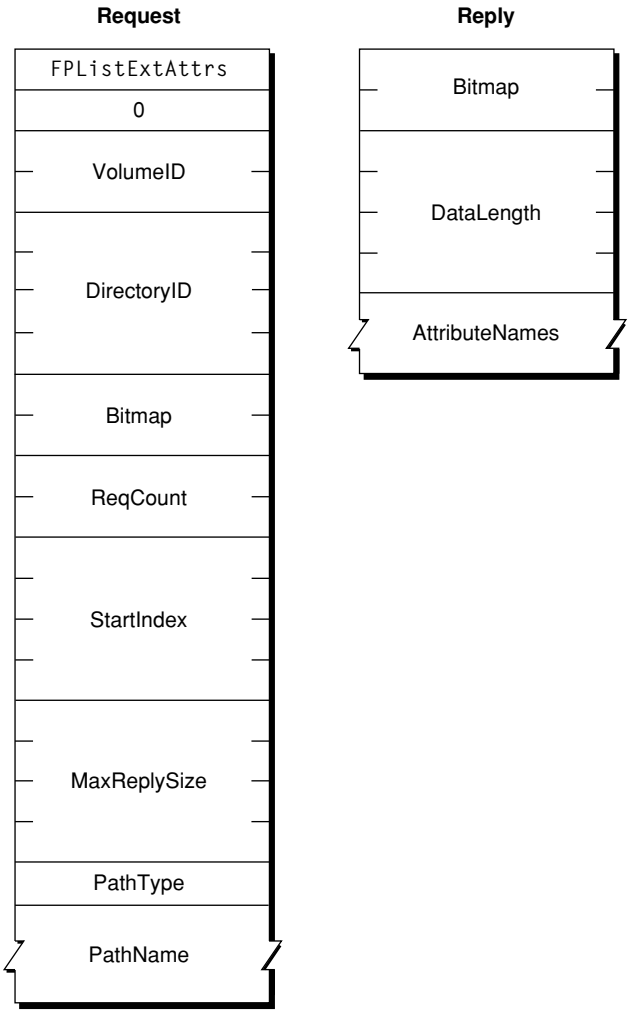
Table 44 describes the reply block for the `FPListExtAttrs` command.

Table 44 Reply block for the `FPListExtAttrs` command

Name and size	Data
`Bitmap` (short)	Reserved.
`DataLength` (unsigned long)	Length of the data that follows. If `MaxReplySize` was set to zero, `DataLength` describes the length of data that would otherwise have been returned.
`AttributeNames` (string)	Series of null-terminated, UTF-8 encoded extended attribute names if `MaxReplySize` was not set to zero.

Figure 37 shows the request and reply blocks for the `FPListExtAttrs` command.

Figure 52 Request and reply blocks for the `FPListExtAttrs` command



Version Notes
Introduced in AFP 3.2.

FPLogin

Establishes a session with a server.

```
byte CommandCode
byte Pad
string AFPVersion
string UAM
UserAuthInfo
```

Parameters

CommandCode

`kFPLogin` (18).

Pad

Pad byte.

AFPVersion

String indicating which AFP version to use. For possible values, see AFP Version Strings (page 170).

UAM

String indicating which UAM to use. For possible values, see AFP UAM Strings (page 171).

UserAuthInfo

UAM-dependent information required to authenticate the user (can be null). The data type of UserAuthInfo depends on the UAM specified by UAM.

Result

kFPNoErr if no error occurred. See Table 45 (page 106) for the possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 46 (page 107) for the format of the reply block.

Discussion

This command establishes an AFP session with an AFP server. Before calling FPLogin, the AFP client should call FPGetAuthMethods (page 74) to obtain the AFP versions and UAMs that the server supports. From the list of AFP versions and UAMs returned by FPGetAuthMethods, the AFP client chooses the highest AFP version and the most secure UAM that the client supports and provides them as the AFPVersion and UAM parameters to the FPLogin command.

If the server returns any result code other than kFPAuthContinue or kFPNoErr, a session has not been established.

For more detailed information about UAMs, see "File Server Security" in the "Introduction" section.

The AFP server keeps a count of log in attempts that is reset to zero after every successful login. For every failed login attempt without a preceding successful login, the count is incremented. When the maximum number of failed login attempts is reached, the user's account is disabled. Any attempts to log in after the account is disabled yield a result code of kFPParamErr, indicating that the user is unknown or that his or her account is disabled. The administrator must enable the user's account again. AFP does not notify the administrator that a user's account has been disabled; the user must notify the administrator by some other means.

Table 45 lists the result codes for the FPLogin command.

Table 45 Result codes for the FPLogin command

Result code	Explanation
kFPAuthContinue	Authentication is not yet complete.
kFPBadUAM	Specified UAM is unknown.
kFPBadVersNum	Server does not support the specified AFP version.
kFPCallNotSupported	Server does not support this command.
kFPMiscErr	User is already authenticated.
kFPNoServer	Server is not responding.

Result code	Explanation
kFPPwdExpiredErr	User's password has expired. User is required to change his or her password. The user is logged on but can only change his or her password or log out.
kFPPwdNeedsChangeErr	User's password needs to be changed. User is required to change his or her password. The user is logged on but can only change his or her password or log out.
kFPServerGoingDown	Server is shutting down.
kFPUserNotAuth	Authentication failed.

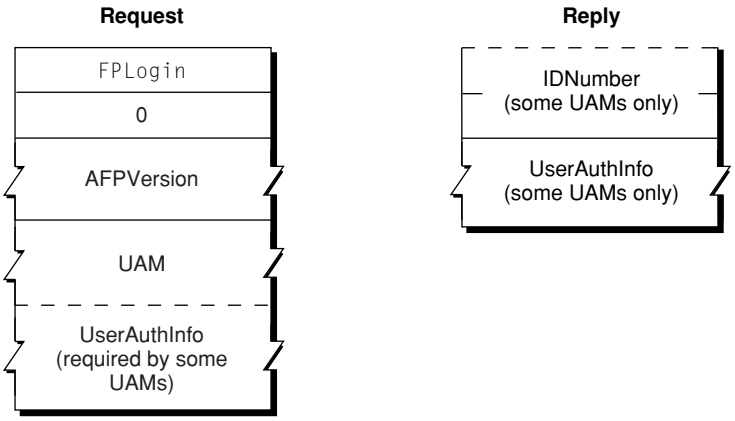
Table 46 describes the reply block for the FPLogincommand.

Table 46 Reply block for the FPLogin command

Name and size	Data
SRefNum (short)	Session reference number used to refer to this session when sending all subsequent commands for this session. The session reference number is valid if kFPNoErr or kFPAuthContinue is returned as the result code.
ID (short)	ID returned by certain UAMs to be passed to the FPLoginCont command. (Valid only when kFPAuthContinue is returned as the result code.)
UserAuthInfo	Value returned by certain UAMs. (Valid only when kFPAuthContinue is returned as the result code.)

Figure 53 shows the request and reply blocks for the FPLogin command.

Figure 53 Request and reply blocks for the FPLogin command



FPLoginCont

Continues the login and user authentication process started by a login command.

```
byte CommandCode
byte Pad
short ID
UserAuthInfo
```

Parameters

CommandCode

kFPLoginCont (19).

Pad

Pad byte.

ID

Number returned by a previous call to FPLogin, FPLoginExt, or FPLoginCont.

UserAuthInfo

UAM-dependent information required to authenticate the user (can be null). The data type of UserAuthInfo depends on the UAM that was specified when FPLogin (page 105) or FPLoginExt (page 109) was called.

Result

kFPNoErr if no error occurred. See Table 47 (page 108) for the possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 48 (page 109) for the format of the reply block.

Discussion

This command sends the ID and UserAuthInfo parameters to the server, which uses them to execute the next step in the UAM. If an additional exchange of packets is required, the server returns a result code of kFPAuthContinue. Otherwise, it returns no kFPNoErr (meaning the user has been authenticated) or kFPUserNotAuth (meaning the authentication has failed). If the server returns no error, SRefNum is valid for use when sending subsequent AFP commands for this session. If the server returns kFPUserNotAuth, it also closes the session and invalidates SRefNum.

If this command returns a result code of kFPPwExpiredErr or kFPPwdNeedsChangeErr, the AFP client should display an explanatory dialog box and allow the user to change his or her password.

Table 47 lists the result codes for the FPLoginCont command.

Table 47 Result codes for the FPLoginCont command

Result code	Explanation
kFPAuthContinue	Authentication is not yet complete.
kFPMiscErr	Non-AFP error occurred.
kFPNoServer	Server is not responding.
kFPParamErr	Authentication failed for an undisclosed reason.

Result code	Explanation
kFPPwdExpiredErr	User's password has expired. User is required to change his or her password. The user is logged in but can only change his or her password or log out.
kFPPwdNeedsChangeErr	User's password needs to be changed. User is required to change his or her password. The user is logged on but can only change his or her password or log out.
kFPUserNotAuth	User was not authenticated because the password is incorrect.

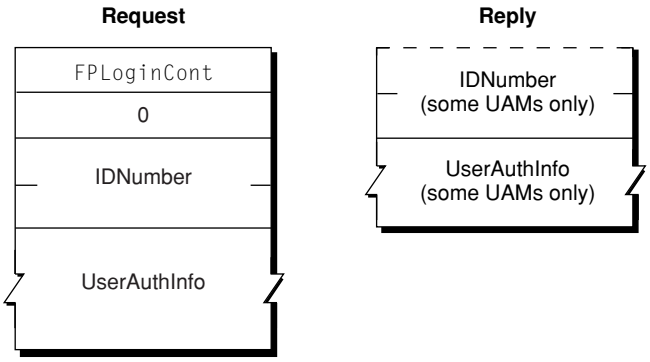
Table 48 describes the reply block for the FPLoginCont command.

Table 48 Reply block for the FPLoginCont command

Name and size	Data
ID (short)	ID returned by certain UAMs to be passed to the FPLoginCont command. (Valid only if kFPAuthContinue is returned as the result code.)
UserAuthInfo	Value returned by certain UAMs. (Valid only if kFPAuthContinue is returned as the result code.)

Figure 54 shows the request and reply blocks for the FPLoginCont command.

Figure 54 Request and reply blocks for the FPLoginCont command



FPLoginExt

Establishes a session with a server using an Open Directory domain.

```
byte CommandCode
byte Pad
short Flags
string AFPVersion
string UAM
byte UserNameType
AFPName UserName
byte PathType
string Pathname
byte Pad
UserAuthInfo
```

Parameters

CommandCode

`kFPLoginExt` (63).

Pad

Pad byte.

Flags

Flags providing additional information. (No flags are currently defined.)

AFPVersion

String indicating which AFP version to use. For possible values, see `AFP Version Strings` (page 170).

UAM

String indicating which UAM to use. For possible values, see `AFP UAM Strings` (page 171).

UserNameType

Type of name in `UserName`; always 3.

UserName

UTF-8–encoded name of the user.

PathType

Type of names in `PathName`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname for the Open Directory domain in which the user specified by `UserName` can be found. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Pad

Pad byte that may be required for `Pathname` to end on an even boundary.

UserAuthInfo

UAM-dependent information required to authenticate the user (can be null). The data type of `UserAuthInfo` is dependent on the UAM specified by `UAM`.

Result

`kFPNoErr` if no error occurred. See Table 49 (page 111) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr` or `kFPAuthContinue`, the server returns a reply block. See Table 50 (page 112) for the format of the reply block.

Discussion
This command establishes an AFP session using the specified Open Directory domain in which information about the user can be found. Before sending this command, the AFP client should call `FPGetAuthMethods` (page 74) to obtain the UAMs that the Open Directory domain supports. From the list of UAMs returned by `FPGetAuthMethods`, the AFP client chooses the most secure UAM that it supports and provides it in the `UAM` parameter of the `FPLoginExt` command.

If the server returns any result code other than `kFPAuthContinue` or `KFPNoErr`, a session has not been established.

For more detailed information about UAMs, see "File Server Security" in the "Introduction" section.

The AFP server keeps a count of log in attempts that is reset to zero after every successful login. For every failed log in attempt without a preceding successful log in, the count is incremented. When the maximum number of failed log in attempts is reached, the user's account is disabled. Any attempts to log in after the account is disabled result in an `kFPParamErr` indicating that the user is unknown or that his or her account is disabled. The administrator must enable the user's account again. AFP does not notify the administrator that a user's account has been disabled; the user must notify the administrator by some other means.

Table 49 lists the result codes for the `FPLoginExt` command.

Table 49 Result codes for the `FPLoginExt` command

Result code	Explanation
`kFPAuthContinue`	Authentication is not yet complete.
`kFPBadUAM`	Specified UAM is unknown.
`kFPBadVersNum`	Server does not support the specified AFP version.
`kFPMiscErr`	User is already authenticated.
`kFPParamErr`	Specified user is unknown or the account has been disabled due to too many login attempts.
`kFPPwdExpiredErr`	User's password has expired. User is required to change his or her password. The user is logged on but can only change his or her password or log out.
`kFPPwdNeedsChangeErr`	User's password needs to be changed. User is required to change his or her password. The user is logged in but can only change his or her password or log out.
`kFPNoServer`	Server is not responding.
`kFPServerGoingDown`	Server is shutting down.
`kFPUserNotAuth`	Authentication failed.

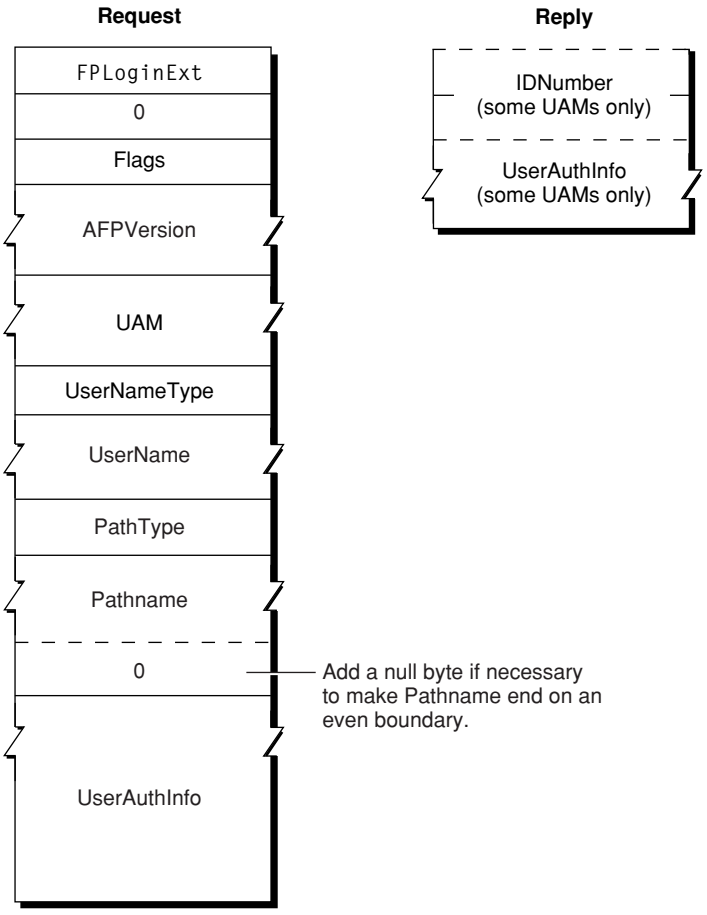
Table 50 describes the reply block for the `FPLoginExt`command.

Table 50 Reply block for the `FPLoginExt` command

Name and size	Data
`SRefNum` (short)	Session reference number used to refer to this session when sending all subsequent commands for this session. The session reference number is valid if `kFPNoErr` or `kFPAuthContinue` is returned as the result code.
`ID` (short)	ID returned by certain UAMs to be passed to the FPLoginCont command. (Valid only when `kFPAuthContinue` is returned as the result code.)
`UserAuthInfo`	Value returned by certain UAMs when `kFPAuthContinue` is returned as the result code.

Figure 55 shows the request and reply blocks for the `FPLoginExt` command.

Figure 55 Request and reply blocks for the `FPLoginExt` command



FPLogout

Terminates a session with a server.

```
byte CommandCode
byte Pad
```

Parameters

CommandCode

`kFPLogout` (20).

Pad

Pad byte.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

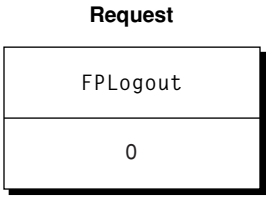
None.

Discussion

This command terminates sessions established by `FPLogin` (page 105)and `FPLoginExt` (page 109). The server flushes and closes any forks opened by the session, frees all session-related resources, and invalidates the session reference number.

Figure 56 shows the request block for the `FPLogout` command.

Figure 56 Request block for the `FPLogout` command



FPMapID

Maps a User ID to a user name or a Group ID to a group name.

```
byte CommandCode
byte Subfunction
long ID
```

Parameters

CommandCode

`kFPMapID` (21).

Subfunction

Subfunction code, where 1 maps a User ID to a Macintosh Roman user name, 2 maps a Group ID to a Macintosh Roman group name, 3 maps a User ID to a UTF-8–encoded user name, and 4 maps a Group ID to a UTF-8–encoded group name.

ID

Group ID or User ID that is to be mapped.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or subfunction code is unknown, kFPItemNotFound if the ID was not found, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. The reply block consists of a string, called Name, containing the name that corresponds to ID. The name can be a string of up to 255 Macintosh Roman characters or an AFPName of up to 255 characters.

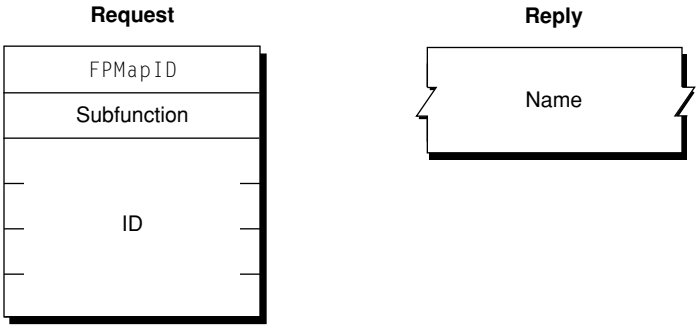
Discussion

The server retrieves the user or group name in that corresponds to the specified User ID or Group ID.

The Subfunction parameter tells the server which database (user or group) to search first. User and group IDs come from the same pool of numbers, so if the ID has been assigned, FPMapID always returns a user or group name.

Figure 57 shows the request and reply blocks for the FPMapID command.

Figure 57 Request and reply blocks for the FPMapID command



FPMapName

Maps a user name to a User ID or a group name to a Group ID.

```
byte CommandCode
byte Subfunction
string Name
```

Parameters

CommandCode

kFPMapName (22).

Subfunction

Subfunction code, where 1 maps a UTF-8–encoded user name to a User ID, 2 maps a UTF-8–encoded group name to a Group ID, 3 maps a Macintosh Roman user name to User ID, and 4 maps a Macintosh Roman group name to a Group ID.

Name

Name that is to be mapped to an ID. The name can be a string of up to 255 Macintosh Roman characters or an AFPName of up to 255 characters.

Result

kFPNoErr if no error occurred, kFPParamErr if the session reference number or subfunction code is unknown, kFPItemNotFound if the ID was not found, or kFPMiscErr if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. The reply block consists of a long, called ID, containing the ID corresponding to the input name.

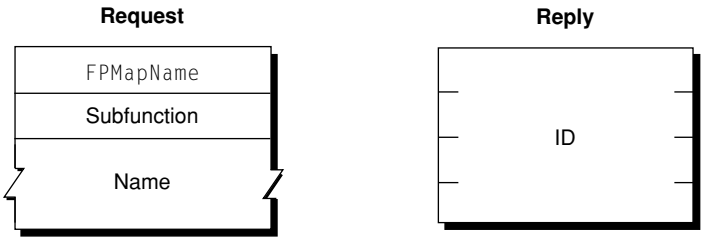
Discussion

The server retrieves the ID number that corresponds to the specified user or group name or returns a kFPItemNotFound result code if it does not find the name in its list of valid names.

The Subfunction parameter tells the server which database (user or group) to search first. If you have a user and a group that are both named "Fred" and you call FPMapName, the subfunction code will determine in which database (user or group) the match is found.

Figure 58 shows the request and reply blocks for the FPMapName command.

Figure 58 Request and reply blocks for the FPMapName command



FPMoveAndRename

Moves a CNode to another location on a volume or renames a CNode.

```
byte CommandCode
byte Pad
short VolumeID
long SourceDirectoryID
long DestDirectoryID
byte SourcePathType
string SourcePathname
byte DestPathType
string DestPathname
byte NewType
string NewName
```

Parameters

CommandCode

kFPMoveAndRename (23).

Pad

Pad byte.

VolumeID

Volume ID.

SourceDirectoryID
Source ancestor Directory ID.

DestDirectoryID
Destination ancestor Directory ID.

SourcePathType
Type of names in SourcePathname. See Path Type Constants (page 174) for possible values.

SourcePathname
Pathname of the file or directory to be moved (may be null if a directory is being moved). SourcePathname is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

DestPathType
Type of names in DestPathname. See Path Type Constants (page 174) for possible values.

DestPathname
Pathname of the file or directory to be moved (may be null if a directory is being moved). DestPathname is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

NewType
Type of name in NewName. See Path Type Constants (page 174) for possible values.

NewName
New name of file or directory (may be null). NewName is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result
kFPNoErr if no error occurred. See Table 51 (page 117) for other possible result codes.

ReplyBlock
If the result code is kFPNoErr, the server returns a reply block. The reply block consists of a long, called ID, containing the ID corresponding to the input name.

Discussion
This command copies and optionally renames the CNode and removes the CNode from the original parent directory. If the NewName parameter is null, the moved CNode retains it original name. Otherwise, the server moves the CNode, creating the Long or Short Names as described in the section "Catalog Node Names" in Chapter 1. The CNode's modification date and the modification date of the source and destination parent directories are set to the server's clock. The CNode's Parent ID is set to the destination Parent ID. All other parameters remain unchanged, and if the CNode is a directory, the parameters of all descendent directories and files remain unchanged.

The FPMoveAndRename command indicates the destination of the move by specifying the ancestor Directory ID and the pathname to the CNode's destination parent directory.

If the CNode being moved is a directory, all its descendents are moved as well.

To move a directory, the user must have search access to all ancestors, down to and including the source and destination parent directories, as well as write access to those directories. To move a file, the user must have search access to all ancestors, except the source and destination parents, as well as read and write access to the source parent directory and write access to the destination parent directory.

A CNode cannot be moved from one volume to another with this command, even if both volumes are managed by the same server.

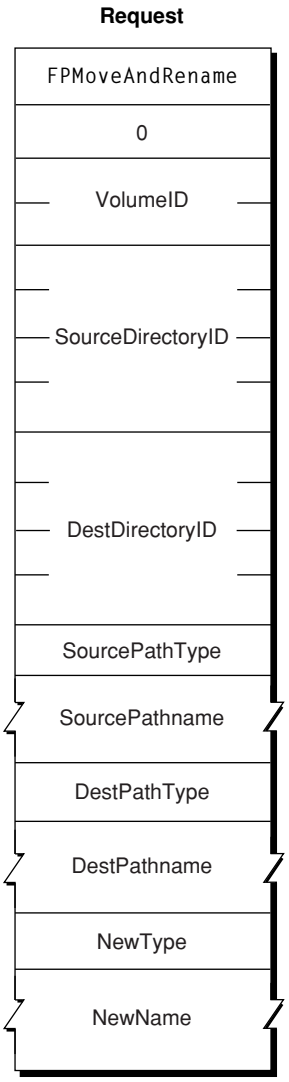
Table 51 lists the result codes for the FPMoveAndRename command.

Table 51 Result codes for the `FPMoveAndRename` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to move or rename the specified file or directory.
`kFPCantMove`	Attempt was made to move a directory into one of its descendent directories.
`kFPInsideSharedErr`	Directory being moved contains a share point and is being moved into a directory that is shared or is the descendent of a directory that is shared.
`kFPInsideTrashErr`	Shared directory is being moved into the Trash; a directory is being moved to the trash and it contains a shared folder.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectExists`	File or directory having the name specified by `NewName` already exists.
`kFPObjectLocked`	Directory being moved, renamed, or moved and renamed is marked RenameInhibit; file being moved and renamed is marked RenameInhibit.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; a pathname or `NewName` is invalid.
`kFPVolLocked`	Volume is ReadOnly.

Figure 59 shows the request block for the `FPMoveAndRename` command.

Figure 59 Request block for the `FPMoveAndRename` command



FPOpenDir

Opens a directory on a variable Directory ID volume and returns its Directory ID.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPOpenDir` (25).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname of the file or directory to be moved (may be null if a directory is being moved). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

`kFPNoErr` if no error occurred. See Table 52 (page 119) for other possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. The reply block consists of a long, called DirectoryID, containing the Directory ID of the opened directory.

Discussion

If `VolumeID` specifies a variable Directory ID volume, the server generates a Directory ID for the specified directory. If VolumeID specifies a fixed Directory ID type, the server returns the fixed Directory ID belonging to the directory specified by `Pathname`.

Although this command can obtain a Directory ID for a directory on a fixed Directory ID volume, the recommended way to obtain a Directory ID for a directory on a fixed Directory ID volume is to call `FPGetFileDirParms` (page 80).

The user must have search access to all ancestors down to and including the specified directory's parent directory.

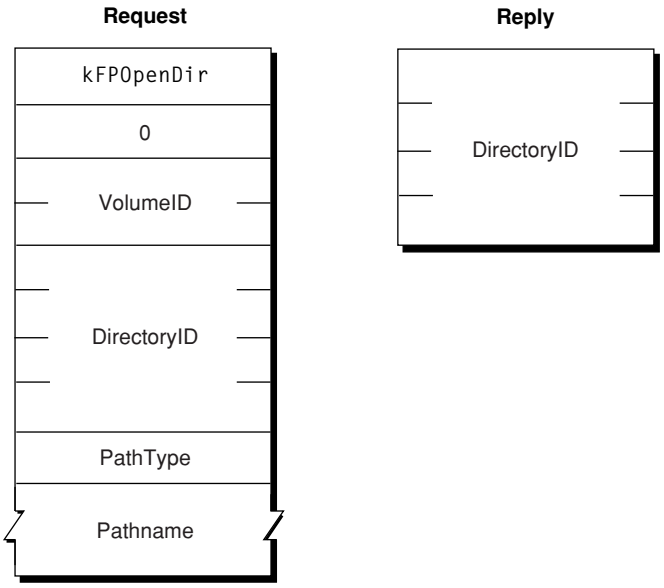
Table 52 lists the result codes for the `FPOpenDir` command.

Table 52 Result codes for the `FPOpenDir` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to open the directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing directory.
`kFPObjectTypeErr`	Input parameters point to a file.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; a pathname is invalid.

Figure 60 shows the request and reply blocks for the `FPOpenDir` command.

Figure 60 Request and reply blocks for the `FPOpenDir` command



FPOpenDT

Opens the Desktop database on a particular volume.

```
byte CommandCode
byte Pad
short VolumeID
```

Parameters

CommandCode

`kFPOpenDT` (48).

Pad

Pad byte.

VolumeID

Volume ID.

Result

`kFPNoErr` if no error occurred, `kFPParamErr` if the session reference number or `VolumeID` is unknown, or `kFPMiscErr` if an error occurred that is not specific to AFP.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. The reply block consists of a short, called `DTRefNum`, containing a Desktop database reference number.

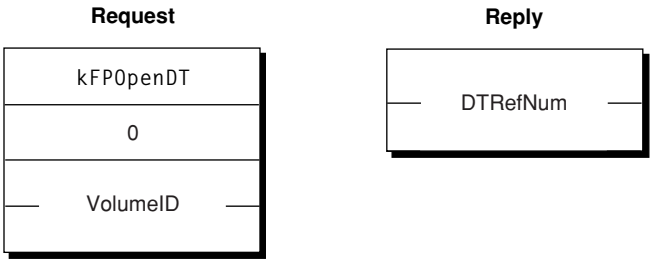
Discussion

The server opens the Desktop database on the selected volume and returns a Desktop database reference number that is unique among such reference numbers. The Desktop database reference number is to be used in all subsequent Desktop database commands relating to this volume.

The user must have previously called `FPOpenVol` (page 124) for this volume.

Figure 61 shows the request and reply blocks for the `FPOpenDT` command.

Figure 61 Request and reply blocks for the `FPOpenDT` command



FPOpenFork

Opens a fork of an existing file for reading or writing.

```
byte CommandCode
byte Flag
short VolumeID
long DirectoryID
short Bitmap
short AccessMode
byte PathType
string Pathname
```

Parameters

CommandCode

`kFPOpenFork` (26).

Flag

Bit 7 of the `Flag` parameter is the `ResourceDataFlag` bit, and it indicates which fork to open, where 0 specifies the data fork and 1 specifies the resource fork.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

Bitmap

Bitmap describing the fork parameters to be returned. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the `FPGetFileDirParms` (page 80) command and can be null. For bit definitions for the File bitmap, see File Bitmap (page 164).

AccessMode

Desired access and deny modes, specified by any combination of the following bits: 0 = Read — allows the fork to be read 1 = Write — allows the fork to be written 4 = DenyRead — prevents others from reading the fork while it is open 5 = DenyWrite — prevents others from writing the fork while it is open For more information on access and deny modes, see “File Sharing Modes” in the “Introduction” section.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired file (cannot be null). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

kFPNoErr if no error occurred. See Table 53 (page 122) for the possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 54 (page 123) for the format of the reply block.

Discussion

The server opens the specified fork if the user has the access rights for the requested access mode and if the access mode does not conflict with already-open access paths to the fork.

If the fork is opened, the server returns in the reply block a copy of the input bitmap, an open fork reference number for use with all subsequent commands involving the opened fork, and followed by file parameters packed in bitmap order.

File parameters are returned only if the command completes without error or if the command returns with a kFPDenyConflict result code. In the latter case, the server returns a fork reference of zero.

A kFPBitmapErr result code is returned if an attempt is made to retrieve the length of the file's other fork.

The server needs to keep variable-length parameters, such as Long Name or Short Name, at the end of the reply block. In order to do this, the server represents variable-length parameters in bitmap order as fixed-length offsets (integer) to the start of the variable-length parameters. The actual variable-length fields are then packed after all fixed-length parameters.

If the fork is opened and the user has requested the file's attributes in the file bitmap, the appropriate DAlreadyOpen or RAlreadyOpen bit is set.

To open a fork for read or no access (when neither read or write access is requested), the user must have search access to all ancestors, except the parent directory, as well as read access to the parent directory. For information about access modes, see "File Sharing Modes" in the "Introduction" section.

To open a fork for write access, the volume must not be designated for read-only access. If both forks are currently empty, the user must have search or write access to all ancestors, except the parent directory, as well as write access to the parent directory. If either fork is not empty and one of the forks is being opened for writing, the user must have search access to all ancestors, except the parent directory, as well as read and write access to the parent directory.

The user must have previously called FPOpenVol (page 124) for this volume. Each fork must be opened separately; a unique fork reference is returned for each fork.

Table 53 lists the result codes for the FPOpenFork command.

Table 53 Result codes for the FPOpenFork command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to open the specified fork.
kFPBitmapErr	Attempt was made to retrieve a parameter that cannot be obtained with this command (the fork is not opened).
kFPDenyConflict	File or fork cannot be opened because of a deny modes conflict.
kFPMiscErr	Non-AFP error occurred.

Result code	Explanation
kFPObjectNotFound	Input parameters do not point to an existing file.
kFPObjectLocked	Attempt was made to open a file for writing that is marked WriteInhibit.
kFPObjectTypeErr	Input parameters point to a directory.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown; a pathname is invalid.
KFPTooManyFilesOpen	Server cannot open another fork.
kFPVolLocked	Attempt was made to open for writing a file on a volume that is marked ReadOnly.

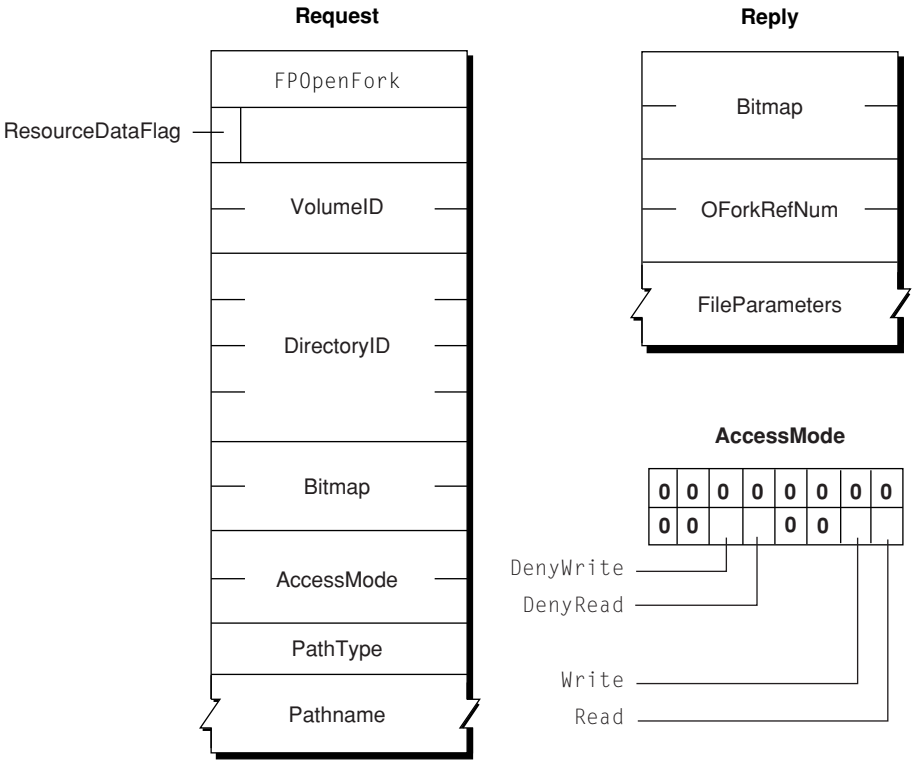
Table 54 describes the reply block for the FPOpenForkcommand.

Table 54 Reply block for the FPOpenFork command

Name and size	Data
Bitmap (short)	Copy of the input parameter.
OForkRefNum (short)	Open fork reference number for use when referring to this fork in when sending subsequent commands.
FileParameters	Requested parameters.

Figure 62 shows the request and reply blocks for the FPOpenFork command.

Figure 62 Request and reply blocks for the `FPOpenFork` command



FPOpenVol

Opens a volume.

```
byte CommandCode
byte Pad
short Bitmap
string VolumeName
8 bytes Password
```

Parameters

CommandCode

`kFPOpenVol` (24).

Pad

Pad byte.

VolumeID

Volume ID.

Bitmap

Bitmap describing the parameters that are to be returned. Set the bit that corresponds to each desired parameter. The bitmap is the same as the Volume bitmap used by the `FPGetVolParms` (page 101) command and cannot be null. For bit definitions, see Volume Bitmap (page 168).

VolumeName

Name of the volume as returned by `FPGetSrvrParms` (page 98).

Password

Optional volume password.

Result

kFPNoErr if no error occurred. See Table 55 (page 125) for the possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 56 (page 125) for the format of the reply block.

Discussion

This command must be made before any other command can be made to obtain access to CNodes on the specified volume.

If a password is required to gain access to the volume, it is sent as the Password parameter in cleartext. Append null bytes to the password as necessary to obtain a length of eight bytes. Password comparison is case-sensitive. If the supplied password does not match the password kept with the volume, or if a password is not supplied when a password is required, the server returns a result code of kFPAccessDenied.

If the passwords match, or if the volume is not password-protected, the server packs the requested parameters in the reply block The user can now send commands related to CNodes on the volume.

The Bitmap parameter must request that the Volume ID be returned. There is no other way to retrieve the Volume ID, which is required by most subsequent commands related to this volume.

Table 55 lists the result codes for the FPOpenVol command.

Table 55 Result codes for the FPOpenVol command

Result code	Explanation
kFPAccessDenied	Password is not supplied or does not match.
kFPBitmapErr	Attempt was made to retrieve a parameter that cannot be obtained with this command. (The bitmap is null.)
kFPMiscErr	Non-AFP error occurred.
kFPObjectNotFound	Input parameters do not point to an existing volume.
kFPParamErr	Session reference number or volume name is unknown.

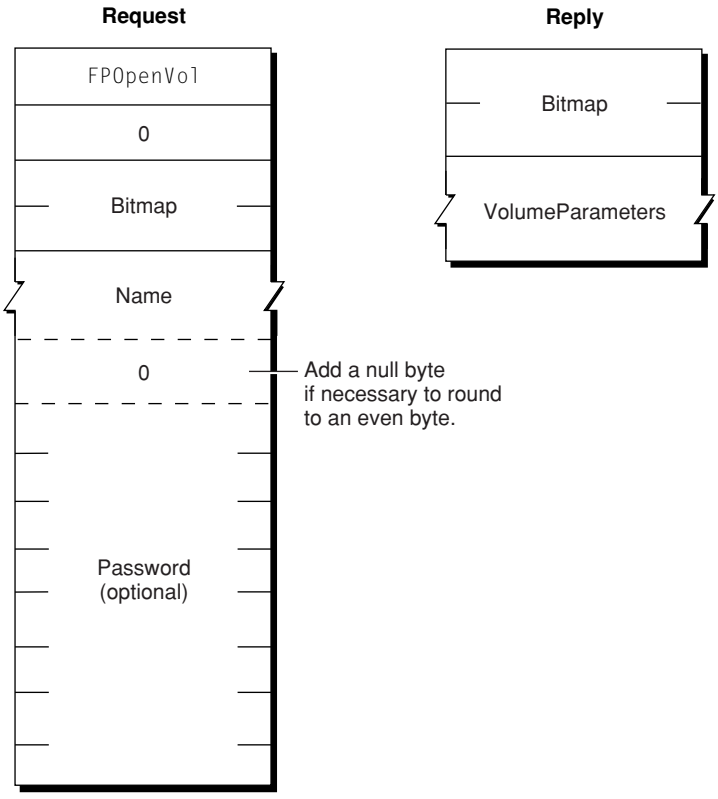
Table 56 describes the reply block for the FPOpenVol command.

Table 56 Reply block for the FPOpenVol command

Name and size	Data
Bitmap (short)	Copy of the input parameter.
VolumeParameters	Requested parameters, including the Volume ID, for the opened volume.

Figure 63 shows the request and reply blocks for the FPOpenVol command.

Figure 63 Request and reply blocks for the `FPOpenVol` command



FPRead

Reads a block of data.

```
byte CommandCode
byte Pad
short OForkRefNum
long Offset
long ReqCount
byte NewLineMask
byte NewLineChar
```

Parameters

CommandCode

`kFPRead` (27).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Offset

Number of the first byte to read.

ReqCount
Number of bytes to read.

NewLineMask
Mask for determining where the read should terminate.

NewLineChar
Character for determining where the read should terminate.

Result
kFPNoErr if no error occurred. See Table 57 (page 127) for the possible result codes.

ActualCount
Number of bytes actually read from the fork. This long value is returned by the underlying transport mechanism and is not a value in the reply block.

ReplyBlock
If the result code is kFPNoErr, the server returns a reply block containing the data that was read.

Discussion
This command retrieves the specified range of bytes from an open fork. Call FPOpenFork (page 121) to open the fork. The server begins reading at the byte number specified by the Offset parameter. Reading stops when one of the following occur:

- The server encounters the character specified by the combination of the NewLineMask and NewLineChar parameters
- The server reaches the end of the fork
- The server encounters the start of a range locked by another user
- The server reads the number of bytes specified by the ReqCount parameter

If the server reaches the end of fork or the start of a locked range, it returns all data read to that point and a result code of kFPEOFErr or kFPLockErr, respectively.

The NewLineMask parameter is a byte mask that is to be logically ANDed with a copy of each byte read. If the result matches the NewLineChar parameter, the read terminates. Using a NewLineMask value of zero essentially disables the Newline check feature.

If a user reads a byte that was never written to the fork, the result is undefined.

Lock the range to be read before sending this command. The underlying transport mechanism may force the request to be broken into multiple smaller requests. If the range is not locked when this command begins execution, it is possible for another user to lock some or all of the range before this command completes, causing the read to succeed partially.

Table 57 (page 127) lists the result codes for the FPRead command.

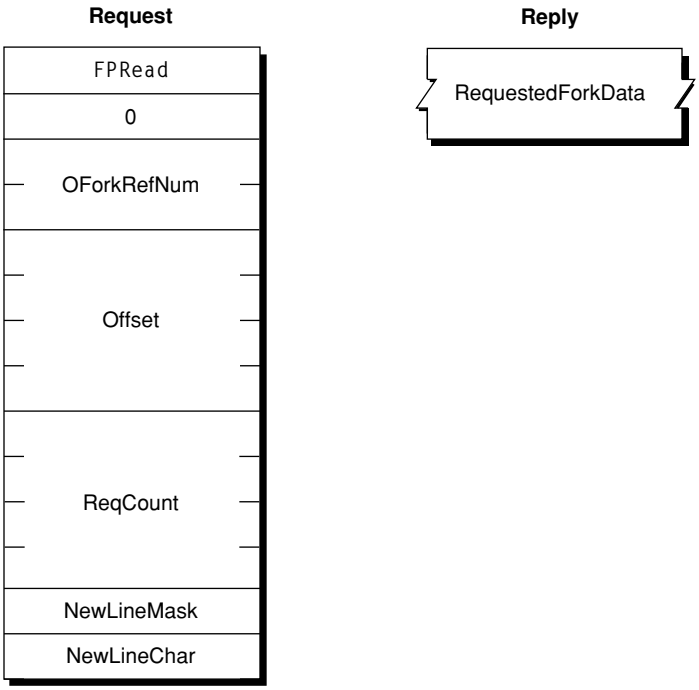
Table 57 Result codes for the FPRead command

Result code	Explanation
kFPAccessDenied	Fork was not opened for read access.
kFPEOFErr	End of fork was reached.
kFPLockErr	Some or all of the requested range is locked by another user.

Result code	Explanation
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or open fork reference number is unknown; ReqCount or Offset is negative; NewLineMask is invalid.

Figure 64 shows the request and reply blocks for the FPRead command.

Figure 64 Request and reply blocks for the FPRead command



FPReadExt

Reads a block of data.

```
byte CommandCode
byte Pad
short OForkRefNum
long long Offset
long long ReqCount
```

Parameters

CommandCode

kFPReadExt (60).

Pad

Pad byte.

OForkRefNum
Open fork reference number.

Offset
Number of the first byte to read.

ReqCount
Number of bytes to read.

Result
`kFPNoErr` if no error occurred. See Table 58 (page 129) for the possible result codes.

ActualCount
Number of bytes actually read from the fork. This long long value is returned by the underlying transport mechanism and is not a value in the reply block.

ReplyBlock
If the result code is `kFPNoErr`, the server returns a reply block containing the data that was read.

Discussion
This command retrieves the specified range of bytes from an open fork. Call `FPOpenFork` (page 121) to open the fork.

This command differs from the `FPRead` (page 126) command in that this command is prepared to handle large values that may be returned for files the reside in volumes larger than 4 GB in size. Also, this command does not support the `NewlineMask` and `NewlineChar` parameters that `FPRead` supports.

The server begins reading at the byte number specified by the `Offset` parameter. Reading stops when one of the following occur:

- The server reaches the end of the fork
- The server encounters the start of a range locked by another user
- The server reads the number of bytes specified by the `ReqCount` parameter

If the server reaches the end of fork or the start of a locked range, it returns all data read to that point and a result code of `kFPEOFErr` or `kFPLockErr`, respectively.

If a user reads a byte that was never written to the fork, the result is undefined.

Lock the range to be read before sending this command. The underlying transport mechanism may force the request to be broken into multiple smaller requests. If the range is not locked when this command begins execution, it is possible for another user to lock some or all of the range before this command completes, causing the read to succeed partially.

Table 58 lists the result codes for the `FPReadExt` command.

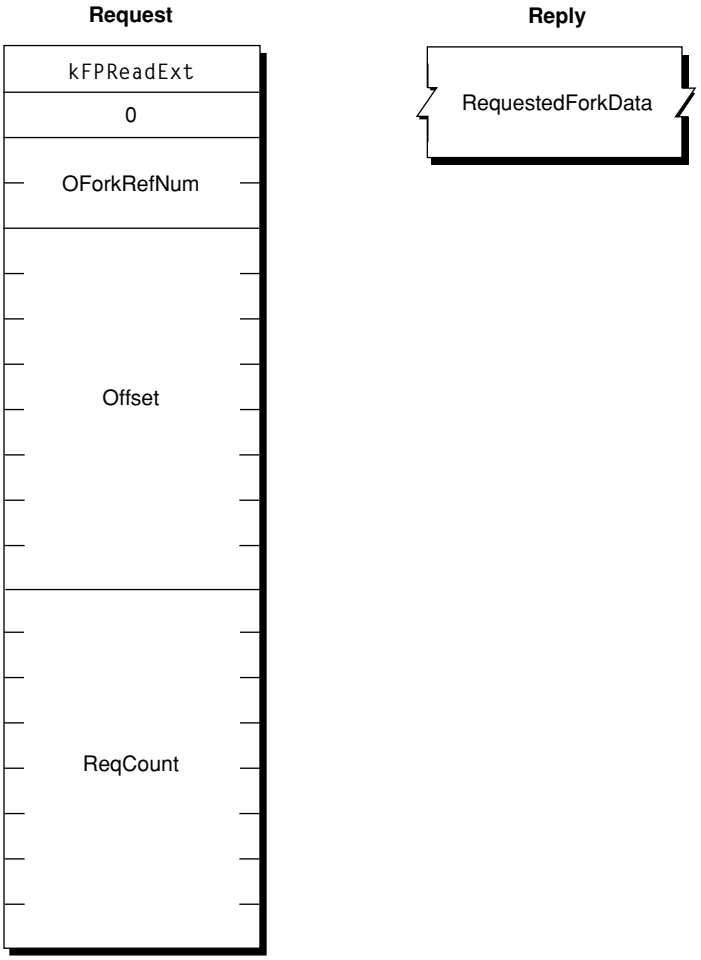
Table 58 Result codes for the `FPReadExt` command

Result code	Explanation
`kFPAccessDenied`	Fork was not opened for read access.
`kFPEOFErr`	End of fork was reached.
`kFPLockErr`	Some or all of the requested range is locked by another user.

Result code	Explanation
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or open fork reference number is unknown; ReqCount or Offset is negative.

Figure 65 shows the request and reply blocks for the FPReadExt command.

Figure 65 Request and reply blocks for the FPReadExt command



FPRemoveAPPL

Removes an APPL mapping from a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long DirectoryID
long FileCreator
byte PathType
string Pathname
```

Parameters

CommandCode

kFPRemoveAPPL (54).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

DirectoryID

Ancestor Directory ID.

FileCreator

File creator of the application corresponding to the APPL mapping that is to be removed.

PathType

Type of names in Pathname. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired file (cannot be null). Pathname is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result

kFPNoErr if no error occurred. See Table 59 (page 131) for the possible result codes.

ReplyBlock

None.

Discussion

The server locates in the Desktop database the APPL mapping corresponding to the specified application and file creator. If an APPL mapping is found, it is removed.

The user must have search access to all ancestors, except the parent directory, as well as read and write access to the parent directory.

The user must have previously called FPOpenDT (page 120) for the corresponding volume. In addition, the file must exist in the specified directory before this command is sent.

Table 59 lists the result codes for the FPRemoveAPPL command.

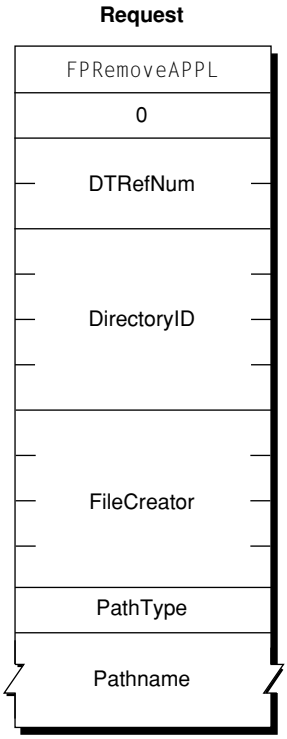
Table 59 Result codes for the FPRemoveAPPL command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.
kFPItemNotFound	No APPL mapping corresponding to the input parameters was found in the Desktop database.
kFPMiscErr	Non-AFP error occurred.

Result code	Explanation
kFPObjectNotFound	Input parameters do not point to an existing file.
kFPParamErr	Session reference number or Desktop database reference number is unknown.

Figure 66 shows the request and reply blocks for the FPRemoveAPPL command.

Figure 66 Request and reply blocks for the FPRemoveAPPL command



FPRemoveComment

Removes a comment from a volume's Desktop database.

```
byte CommandCode
byte Pad
short DTRefNum
long DirectoryID
byte PathType
string Pathname
```

Parameters

CommandCode

kFPRemoveComment (57).

Pad

Pad byte.

DTRefNum
Desktop database reference number.

DirectoryID
Ancestor Directory ID.

PathType
Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname
Pathname to the CNode whose comment is being removed (cannot be null). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result
`kFPNoErr` if no error occurred. See Table 60 (page 133) for the possible result codes.

ReplyBlock
None.

Discussion
If the comment is associated with directory that is not empty, the user must have search access to all ancestors, including the parent directory, plus write access to the parent directory. If the comment is associated with an empty directory, the user must have search or write access to all ancestors, including the parent directory, plus write access to the parent directory.

If the comment is associated with a file that is not empty, the user must have search access to all ancestors, except the parent directory, plus read and write access to the parent directory. If the comment is associated with an empty file, the user must have search or write access to all ancestors, except the parent directory, plus write access to the parent directory.

The user must have previously called `FPOpenDT` (page 120) for the corresponding volume.

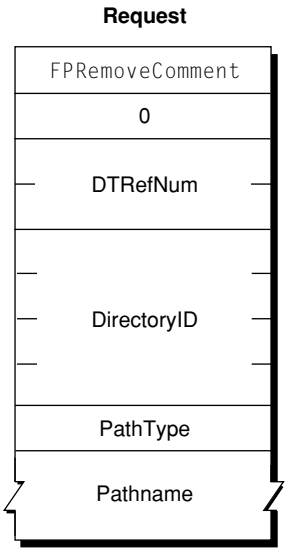
Table 60 lists the result codes for the`FPRemoveComment` command.

Table 60 Result codes for the `FPRemoveComment` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPItemNotFound`	Comment was not found in the Desktop database.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	Session reference number, Desktop database reference number, or pathname type is unknown; pathname is invalid.

Figure 67 shows the request and reply blocks for the `FPRemoveComment` command.

Figure 67 Request and reply blocks for the `FPRemoveComment` command



FPRemoveExtAttr

Removes an extended attribute.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID unsigned short Bitmap
byte PathType
string Pathname
byte Pad
unsigned short NameLength
string Name
```

Parameters

CommandCode

`kFPRemoveExtAttr` (71).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Bitmap specifying the desired behavior when removing an extended attribute. For this command, `kAttrDontFollow` is the only valid bit. For details, see `Extended Attributes Bitmap` (page 164).

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Pad

Optional pad byte if needed to pad to an even boundary.

NameLength

Length in bytes of the extended attribute name that follows.

Name

UTF-8–encoded name of the extended attribute that is to be removed.

Result

`kFPNoErr` if no error occurred. See Table 61 for other possible result codes.

Discussion
This command removes the specified extended attribute.

Support for this command, as well as `FPGetExtAttr` (page 77), `FPListExtAttrs` (page 103), and `FPSetExtAttr` (page 145) is required in order to support extended attributes. UTF-8 support is also required in order to support extended attributes.

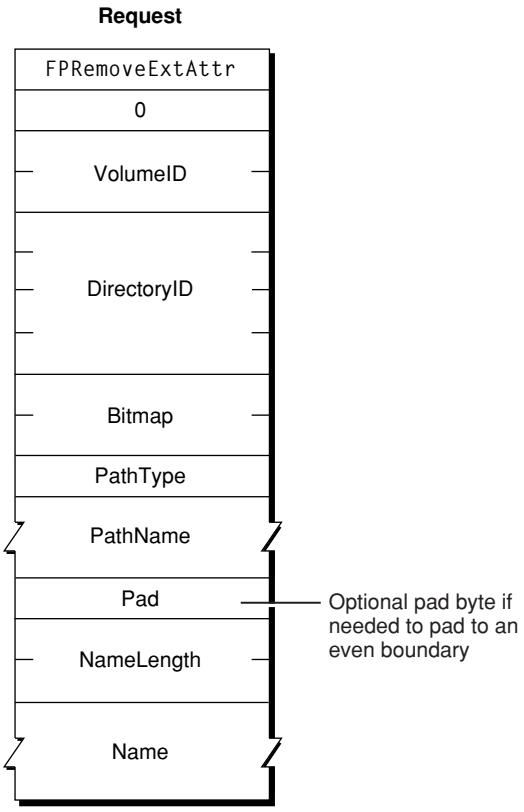
Table 61 lists the possible result codes for the `FPRemoveExtAttr` command.

Table 61 Result codes for the `FPRemoveExtAttr` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to remove an extended attribute for the specified file or directory.
`kFPBitmapErr`	Bitmap is null or specifies a value that is invalid for this command.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

Figure 68 shows the request block for the `FPRemoveExtAttr` command.

Figure 68 Request block for the `FPRemoveExtAttr` command



Version Notes
Introduced in AFP 3.2.

FPRename

Renames a file or directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
byte PathType
string Pathname
byte NewType
string NewName
```

Parameters

CommandCode

`kFPRename` (28).

Pad

Pad byte.

VolumeID
Volume ID.

DirectoryID
Ancestor Directory ID.

PathType
Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname
Pathname to the CNode whose name is being changed (cannot be null). `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

NewType
Type of names in `NewName`. See Path Type Constants (page 174) for possible values.

NewName
Pathname to the CNode, including its new name (cannot be null). `NewName` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Result
`kFPNoErr` if no error occurred. See Table 62 (page 137) for the possible result codes.

ReplyBlock
None.

Discussion
The server assigns the new name to the file or directory. The other name (Long or Short) is generated as described in the section "Catalog Node Names" in Chapter 1. The modification date of the parent directory is set to the server's clock.

To rename a directory, the user must have search access to all ancestors. including the CNode's parent directory, as well as write access to the parent directory. To rename a file, the user must have search access to all ancestors, except the CNode's parent directory, as well as read and write access to the parent directory.

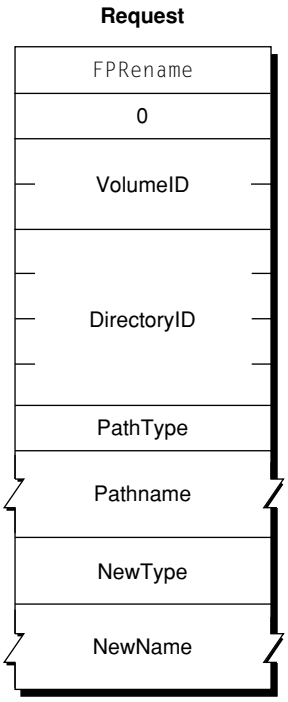
Table 62 lists the result codes for the `FPRename` command.

Table 62 Result codes for the `FPRename` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPCantRename`	Attempt was made to rename a volume or root directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectExists`	File or directory having the name specified by `NewName` already exists.
`kFPObjectLocked`	File or directory is marked RenameInhibit.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname or `NewName` is invalid.
`kFPVolLocked`	Volume is ReadOnly

Figure 69 shows the request block for the `FPRename` command.

Figure 69 Request block for the `FPRename` command



FPResolveID

Gets parameters for a file by File ID.

```
byte CommandCode
byte Pad
short VolumeID
long FileID
short Bitmap
```

Parameters

CommandCode

`kFPResolveID` (41).

Pad

Pad byte.

VolumeID

Volume ID.

FileID

File ID to be resolved.

Bitmap

Bitmap describing the parameters to return. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command. For bit definitions for the this bitmap, see File Bitmap (page 164).

Result

`kFPNoErr` if no error occurred. See Table 63 (page 139) for the possible result codes.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block. See Table 64 (page 139) for the format of the reply block.

Discussion

The parameters returned by this command can be any parameter specified in the `FPGetFileDirParms` (page 80) command.

The user must have the Read Only or the Read & Write privilege to use this command.

Table 63 lists the result codes for the `FPResolveID` command.

Table 63 Result codes for the `FPResolveID` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBadIDErr`	File ID is not valid.
`kFPCallNotSupported`	Server does not support this command.
`kFPIDNotFound`	File ID was not found. (No file thread exists.)
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectTypeErr`	Object defined was a directory, not a file.
`kFPParamErr`	Session reference number, Volume ID, or File ID is unknown.

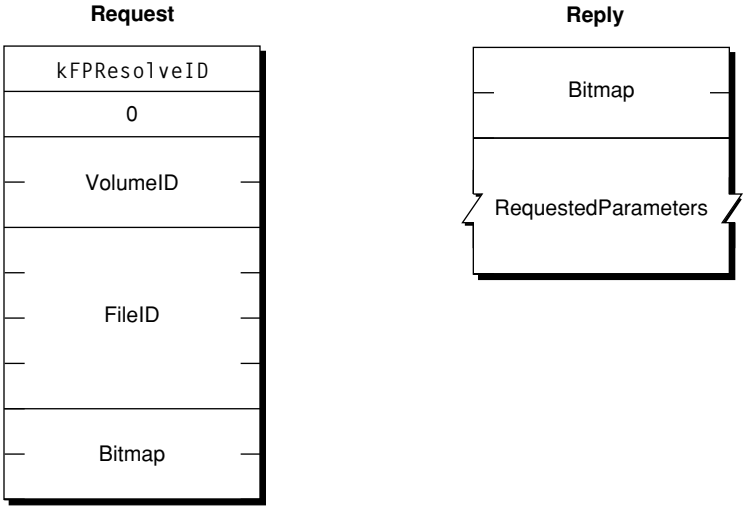
Table 64 describes the reply block for the `FPResolveID` command.

Table 64 Reply block for the `FPResolveID` command

Name and size	Data
`Bitmap` (short)	Copy of the input bitmap.
`FileParameters`	Requested file parameters.

Figure 69 shows the request and reply blocks for the `FPResolveID` command.

Figure 70 Request and reply blocks for the `FPResolveID` command



FPSetACL

Sets the UUID, Group UUID, and ACL for a file or directory and removes an ACL from a file or directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
unsigned short Bitmap
byte Pathtype
string Pathname
byte Pad
AdditionalInformation
```

Parameters

CommandCode

`kFPSetACL` (74).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Bits that specify the values that are to be set. Specify `kFileSec_UUID` to set the UUID of the specified file or directory. Specify `kFileSec_GRPUUID` to set the Group UUID of the specified file or directory. Specify `kFileSec_ACL` to set the ACL of the specified file or directory or `kFileSec_REMOVEACL` to remove the file or directory's ACL. If sending this command is part of the creation of a new item, set the `kFileSec_Inherit` bit. When the server receives an `FPSetACL` command having a `Bitmap` parameter in which the `kFileSec_Inherit` bit is set, it scans the current item looking for access control entries (ACEs) in which the `KAUTH_ACE_INHERITED` bit is set in its `ace_flags` field. The server copies any currently inherited ACEs to the end of the incoming list of ACEs and sets the ACL on the item. For declarations of these constants, see Access Control List Bitmap (page 169).

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname of the Open Directory domain for which UAMs are to be obtained. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Pad

Pad byte if needed to pad to an even boundary.

AdditionalInformation

If `kFileSec_UUID` is set in the `Bitmap` parameter, the first item in this parameter is the UUID that is to be set. If `kFileSec_GRPUUID` is set, the next item in this parameter is the Group UUID that is to be set. If `kFileSec_ACL` is set, the next item in this parameter is a `kauth_acl` structure. For information on this structure, see the section Access Control List Structure (page 161). If `kFileSec_REMOVEACL` is set in the `Bitmap` parameter, this parameter does not contain a `kauth_acl` structure.

Result

`kFPNoErr` if no error occurred. See Table 65 for other possible result codes.

Discussion

Depending on the bits that are set in the `Bitmap` parameter, this command sets the UUID, Group UUID, and ACL for the specified file or directory or removes the ACL of the specified file or directory.

Support for this command, as well as `FPAccess` (page 11) and `FPGetACL` (page 70) is required in order to support access control lists (ACLs). Support for UTF-8 and UUIDs is also required in order to support ACLs.

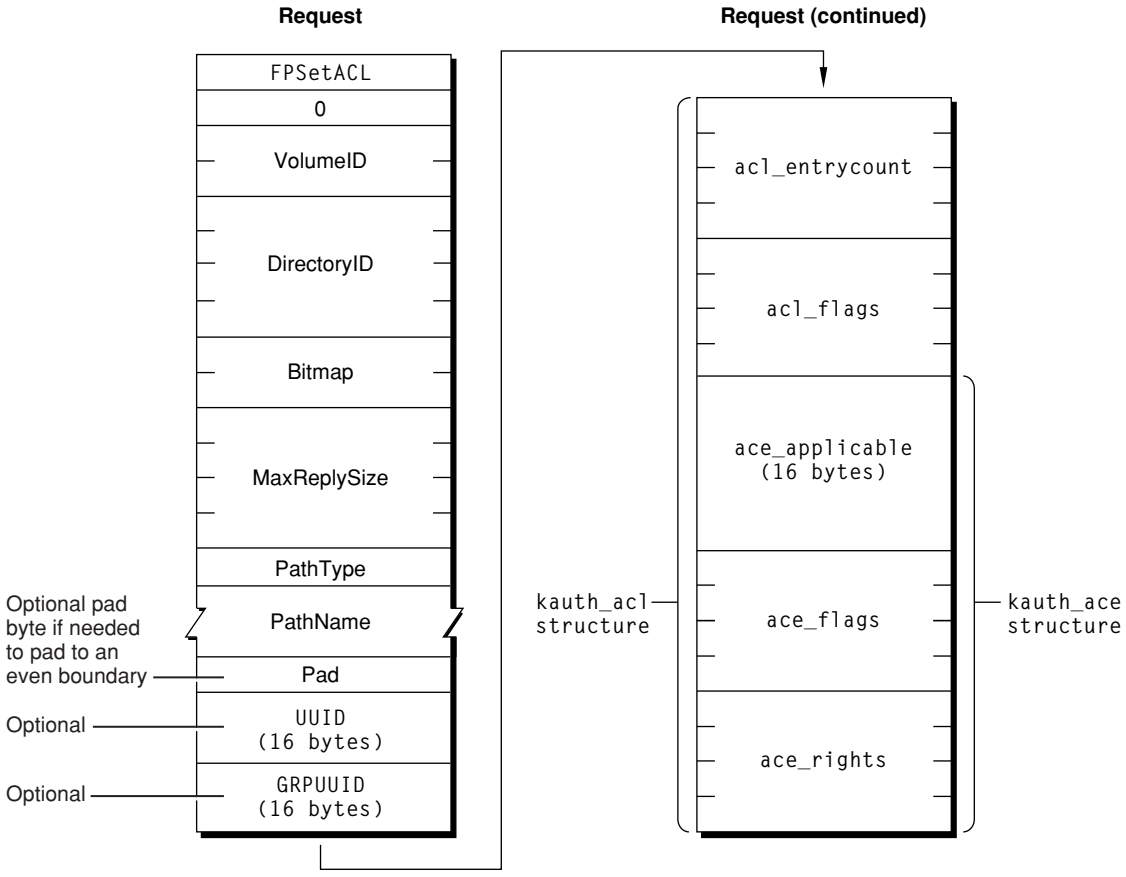
Table 65 lists the result codes for the `FPSetACL` command.

Table 65 Result codes for the `FPSetACL` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access rights required to set the ACL for the specified file or directory.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

Figure 35 shows the request block for the `FPSetACL` command.

Figure 71 Request block for the `FPSetACL` command



Version Notes
Introduced in AFP 3.2.

FPSetDirParms

Sets parameters for a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short Bitmap
byte PathType
string Pathname
DirectoryParameters
```

Parameters

CommandCode

`kFPSetDirParms` (29).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

Bitmap

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 80) command. For bit definitions for this bitmap, see Directory Bitmap (page 162).

PathType

Type of name in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

DirectoryParameters

Parameters to be set, packed in bitmap order.

Result

`kFPNoErr` if no error occurred. See Table 66 (page 144) for the possible result codes.

ReplyBlock

None.

Discussion

This command sets or clears certain parameters and attributes that are common to both files and directories. The parameters are the Invisible and System attributes, Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters, such as Long Name and Short Name, must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

Changing a directory's access rights immediately affects other open sessions. If the user does not have the access rights to set one of the parameters, a `kFPAccessDenied` result code is returned and no parameters are set.

To set a directory's access privileges, Owner ID, Group ID, or to change the DeleteInhibit, RenameInhibit, WriteInhibit, or Invisible attributes, the user must have search or write access to all ancestors, including this directory's parent directory, and the user must be the owner of the directory. To set any parameter other than the ones mentioned above for an empty directory, the user must have search or write access to all ancestors, except the parent directory, as well as write access to the parent directory. To set any parameter other than the ones mentioned above for a directory that is not empty, the user must have search access to all ancestors, including the parent directory, as well as write access to the parent directory.

This command cannot be used to set a directory's name; instead, use `FPRename` (page 136). This command cannot be used to set a directory's Parent Directory ID; instead, use `FPMoveAndRename` (page 115). This command cannot be used to set a directory's Directory ID or Offspring Count.

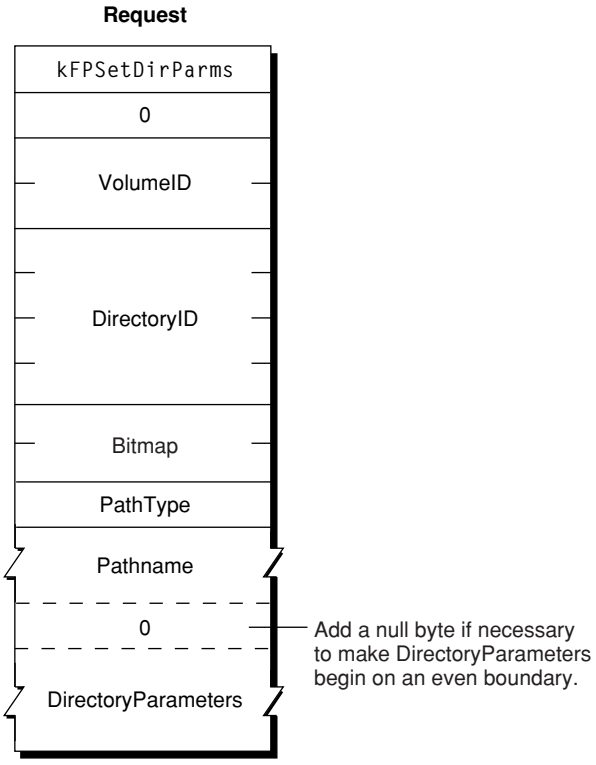
Table 66 lists the result codes for the `FPSetDirParms` command.

Table 66 Result codes for the `FPSetDirParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing directory.
`kFPObjectTypeErr`	Input parameters point to a file.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname, Owner ID, or Group ID is invalid.
`kFPVolLocked`	Volume is ReadOnly.

Figure 72 shows the request block for the `FPSetDirParms` command.

Figure 72 Request block for the `FPSetDirParms` command



FPSetExtAttr

Sets the value of an extended attribute.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID unsigned short Bitmap
long long Offset
byte PathType
string Pathname
byte Pad
unsigned short NameLength
string Name
unsigned long AttributeDataLength
string AttributeData
```

Parameters

CommandCode

`kFPSetExtAttr` (70).

Pad

Pad byte.

VolumeID

Volume identifier.

DirectoryID

Directory identifier.

Bitmap

Bitmap specifying the desired behavior when setting the value of an extended attribute. For details, see `Extended Attributes Bitmap` (page 164) for details.

Offset

Always zero; reserved for future use.

PathType

Type of names in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

Pad

Optional pad byte if needed to pad to an even boundary.

NameLength

Length in bytes of the extended attribute name that follows.

Name

UTF-8–encoded name of the extended attribute that is to be set.

AttributeDataLength

Length in bytes of the extended attribute data that follows.

AttributeData

Value to which the extended attribute is to be set.

Result

`kFPNoErr` if no error occurred. See Table 67 for other possible result codes.

Discussion
This command sets the value of the specified extended attribute. If the extended attribute does not already exist, it is created.

Support for this command, as well as `FPGetExtAttr` (page 77), `FPListExtAttrs` (page 103) and `FPRemoveExtAttr` (page 134) is required in order to support extended attributes. UTF-8 support is also required in order to support extended attributes.

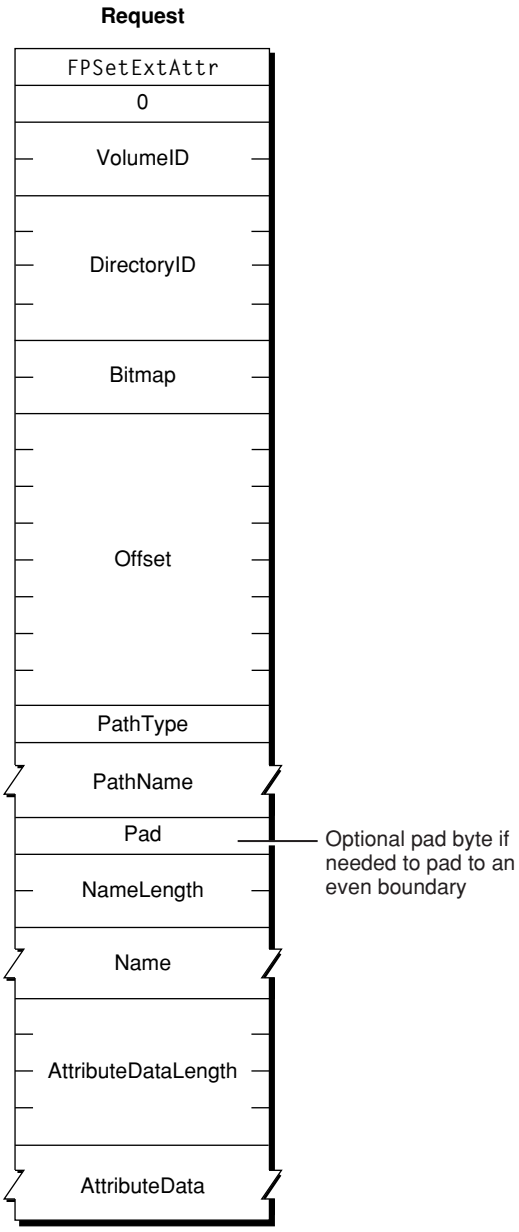
Table 67 lists the possible result codes for the `FPSetExtAttr` command.

Table 67 Result codes for the `FPSetExtAttr` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to set an extended attribute for the file or directory.
`kFPBitmapErr`	Bitmap is null or specifies a value that is invalid for this command.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	A parameter is invalid.

Figure 73 shows the request block for the `FPSetExtAttr` command.

Figure 73 Request block for the `FPSetExtAttr` command



Version Notes
Introduced in AFP 3.2.

FPSetFileDirParms

Sets parameters for a file or a directory.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short Bitmap
byte PathType
string Pathname
FileDirParameters
```

Parameters

CommandCode

`kFPSetFileDirParms` (35).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

Bitmap

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap can be the same as the `DirectoryBitmap` or the `FileBitmap` parameter of the `FPGetFileDirParms` (page 80) command, but this command can only set the parameters that are common to both bitmaps. For bit definitions for the Directory and bitmap, see Directory Bitmap (page 162); for bit definitions for the File bitmap, see File Bitmap (page 164).

PathType

Type of name in `Pathname`. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired file or directory. `Pathname` is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

FileDirParameters

Parameters to be set, packed in bitmap order.

Result

`kFPNoErr` if no error occurred. See Table 68 (page 149) for the possible result codes.

ReplyBlock

None.

Discussion

This command sets or clears certain parameters and attributes that are common to both files and directories. The parameters are the Invisible and System attributes, Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters, such as Long Name and Short Name, must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

If necessary, a null byte must be added between `Pathname` and `DirectoryParameters` in the request block to make `DirectoryParameters` begin on an even boundary.

If the Attributes parameter is included, the Set/Clear bit indicates that the specified attributes are to be set (1) or cleared (0). Therefore, it is not possible to set some attributes and clear other attributes in the same command.

If this command changes the CNode's attributes or sets the CNode's dates (except modification date), Finder Info, or UNIX privileges, the modification date of the CNode is set to the server's clock. If this command changes the CNode's Invisible attribute, the modification date of the CNode's parent directory is set to the server's clock.

To set the parameters for a directory that is not empty, the user needs search access to all ancestors, including the parent directory, as well as write access to the parent directory. To set parameters for an empty directory, the user needs search or write access to all ancestors, except the parent directory, as well as write access to the parent directory.

To set parameters for a file that is not empty, the user needs search access to all ancestors, except the parent directory, as well as write access to the parent directory. To set parameters for an empty file, the user needs search or write access to all ancestors, except the parent directory, as well as write access to the parent directory.

For files, call `FPSetFileParms` (page 150) to set parameters and attributes that `FPSetFileDirParms` cannot set. For directories, call `FPSetDirParms` (page 142) to set parameters and attributes that `FPSetFileDirParms` cannot set.

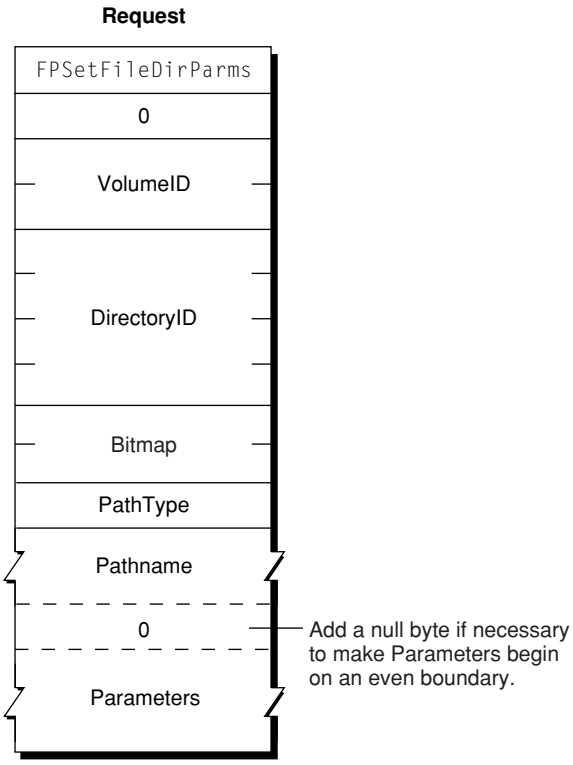
Table 68 lists the result codes for the `FPSetFileDirParms` command.

Table 68 Result codes for the `FPSetFileDirParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file or directory.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is invalid.
`kFPVolLocked`	Volume is ReadOnly.

Figure 74 shows the request block for the `FPSetFileDirParms` command.

Figure 74 Request block for the `FPSetFileDirParms` command



FPSetFileParms

Sets parameters for a file.

```
byte CommandCode
byte Pad
short VolumeID
long DirectoryID
short Bitmap
byte PathType
string Pathname
FileParameters
```

Parameters

CommandCode

`kFPSetFileParms` (30).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

Bitmap

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap can be the same as the FileBitmap parameter of the FPGetFileDirParms (page 80) command. For bit definitions for the Directory bitmap, see Directory Bitmap (page 162); for bit definitions for the File bitmap, see File Bitmap (page 164).

PathType

Type of name in Pathname. See Path Type Constants (page 174) for possible values.

Pathname

Pathname to the desired file or directory. Pathname is a string if it contains Short or Long Names or an AFPName if it contains a UTF-8–encoded path.

FileParameters

Parameters to be set, packed in bitmap order.

Result

kFPNoErr if no error occurred. See Table 69 (page 152) for the possible result codes.

ReplyBlock

None.

Discussion

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

If necessary, a null byte must be added between Pathname and FileParameters in the request block to make FileParameters begin on an even boundary.

The following parameters may be set or cleared: Attributes (all attributes except DAlreadyOpen, RAlreadyOpen, and CopyProtect), Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

If the Attributes parameter is included, the Set/Clear bit indicates that the specified attributes are to be set (1) or cleared (0). Therefore, it is not possible to set some attributes and clear other attributes in the same call.

If this command changes the file's Invisible attribute, the modification date of the file's parent directory is set to the server's clock. If this command changes the file's Attributes or sets any dates (except modification date), or Finder Info, the file's modification date is set to the server's clock.

If the file is empty (both forks are zero length), the user must have search or write access to all ancestors, except this file's parent directory, as well as write access to the parent directory. If either fork is not empty, the user must have search access to all ancestors except the parent directory, as well as read and write access to the parent directory.

This command cannot be used to set a file's name; instead, use FPRename (page 136). This command cannot be used to set the file's Parent Directory ID; instead, use FPMoveAndRename (page 115). This command cannot be used to set a file's fork lengths; instead, call FPSetForkParms (page 152). This command cannot be used to set a file's Node ID.

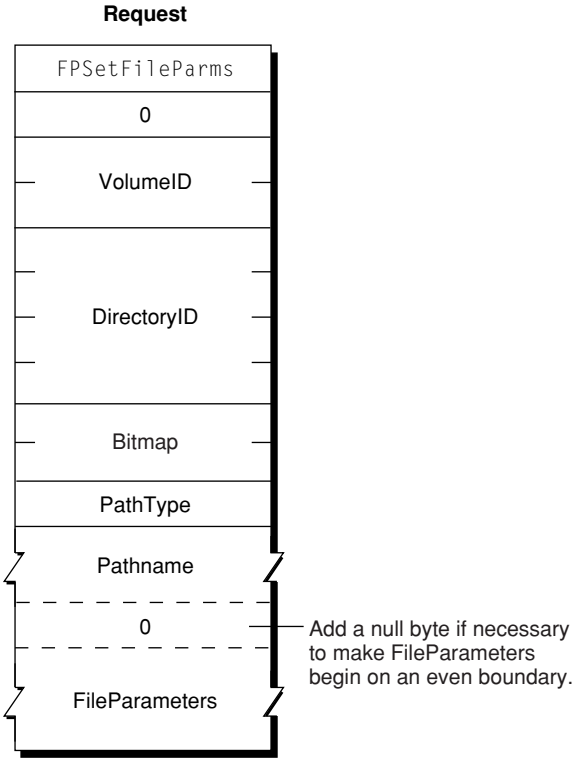
Table 69 lists the result codes for the FPSetFileParms command.

Table 69 Result codes for the `FPSetFileParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file.
`kFPObjectTypeErr`	Input parameters point to a directory.
`kFPParamErr`	Session reference number, Volume ID, or pathname type is unknown; pathname is invalid or null.

Figure 75 shows the request block for the `FPSetFileParms` command.

Figure 75 Request block for the `FPSetFileParms` command



FPSetForkParms

Sets the length of a fork.

```
byte CommandCode
byte Pad
short OForkRefNum
short Bitmap
long ForkLen
```

Parameters

CommandCode

kFPSetForkParms (31).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Bitmap

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap of the FPGetFileDirParms (page 80) command, but only the Data Fork Length, Resource Fork Length, Extended Data Fork Length, and Extended Resource Fork Length parameters can be set. For bit definitions for this bitmap, see File Bitmap (page 164).

ForkLen

New end-of-fork value.

Result

kFPNoErr if no error occurred. See Table 70 (page 153) for the possible result codes.

ReplyBlock

None.

Discussion

The Bitmap and ForkLen parameters are passed to the server, which changes the length of the fork specified by OForkRefNum. The server returns a kFPBitmapErr result code if the command tries to set the length of the file's other fork or if it tries to set any other file parameter.

The server returns a kFPLockErr result code if an attempt is made to truncate the fork in a way that would eliminate a range or part of a range that is locked by another user.

The fork must be open for writing by the user.

This command cannot set a file's name; instead, use FPRename (page 136). This command cannot set a file's Parent Directory ID; instead, use FPMoveAndRename (page 115). This command cannot set a file's file number.

Table 70 lists the result codes for the FPSetForkParms command.

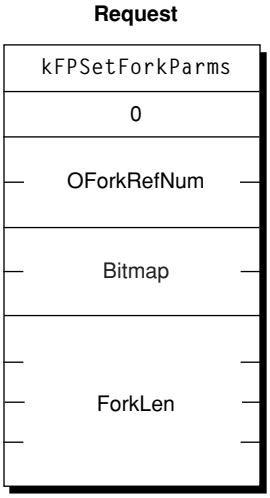
Table 70 Result codes for the FPSetForkParms command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.
kFPBitmapErr	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
kFPDiskFull	No more space exists on the volume.
kFPLockErr	Range lock conflict exists.

Result code	Explanation
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or fork reference number is invalid.
kFPVolLocked	Volume is ReadOnly.

Figure 76 shows the request block for the FPSetForkParms command.

Figure 76 Request block for the FPSetForkParms command



FPSetVolParms

Sets a volume's backup date.

```
byte CommandCode
byte Pad
short VolumeID
short Bitmap
Date BackupDate
```

Parameters

CommandCode

kFPSetVolParms (32).

Pad

Pad byte.

VolumeID

Volume ID.

Bitmap

Bitmap describing the parameters to be set. This parameter is the same as the Bitmap parameter for the FPGetVolParms (page 101) command, but only the Backup Date bit can be set. For bit definitions for this bitmap, see Volume Bitmap (page 168).

BackupDate
New backup date.

Result
`kFPNoErr` if no error occurred. See Table 71 for the possible result codes.

ReplyBlock
None.

Discussion
This command sets a volume's backup date.

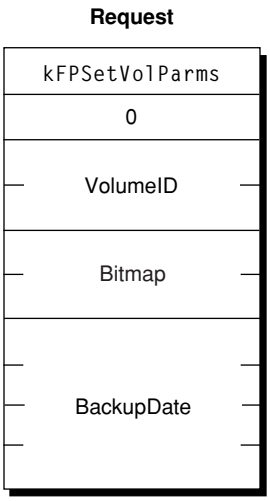
Table 71 lists the result codes for the `FPSetVolParms` command.

Table 71 Result codes for the `FPSetVolParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number is unknown.
`kFPVolLocked`	Volume is ReadOnly.

Figure 77 shows the request block for the `FPSetVolParms` command.

Figure 77 Request block for the `FPSetVolParms` command



FPWrite

Writes a block of data to an open fork.

```
byte CommandCode
byte Flag
short OForkRefNum
long Offset
long ReqCount
ForkData
```

Parameters

CommandCode

kFPWrite (33).

Flag

Bit 7 is the StartEndFlag bit, and it indicates whether Offset is relative to the beginning or end of the fork. A value of zero indicates that the start is relative to the beginning of the fork; a value of 1 indicates that the start is relative to the end of the fork.

OForkRefNum

Open fork reference number.

Offset

Byte offset from the beginning or the end of the fork indicating where the write is to begin; a negative value indicates a byte within the fork relative to the end of the fork.

ReqCount

Number of bytes to be written.

ForkData

Data to be written, which is not a part of the request block. Instead, the data is transmitted to the server in an intermediate exchange of DSI packets.

Result

kFPNoErr if no error occurred. See Table 72 (page 157) for the possible result codes.

ActualCount

Number of bytes actually written to the fork. This long value is returned by the underlying transport mechanism and is not a value in the reply block.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block consisting of a long, called LastWritten, containing the number of the byte just past the last byte written.

Discussion

The server writes data to the open fork, starting at the number of bytes from the beginning or end of the fork as specified by Offset. The StartEndFlag bit indicates whether the block of data is to be written at an offset relative to the beginning or the end of the fork. When the offset is relative to the end of the fork, data can be written without knowing the exact end of the fork, which is useful when multiple writers modify a fork concurrently. The server returns the number of the byte just past the last byte written.

This command differs from the FPWriteExt (page 157) command in that the FPWriteExt command is prepared to handle the large values that may be required for writing to files that reside in volumes larger than 4 GB in size.

If the block of data to be written extends beyond the end of the fork, the fork is extended. If part of the range is locked by another user, the server returns a kFPLockErr result code and does not write any data to the fork.

The file's Modification Date is not changed until the fork is closed.

The fork must be open for writing by the user sending this command.

Lock the range before sending this command. The underlying transport mechanism may force the request to be broken into multiple smaller requests. If the range is not locked when this command begins execution, it is possible for another user to lock some or all of the range before this command completes, causing the write to succeed partially.

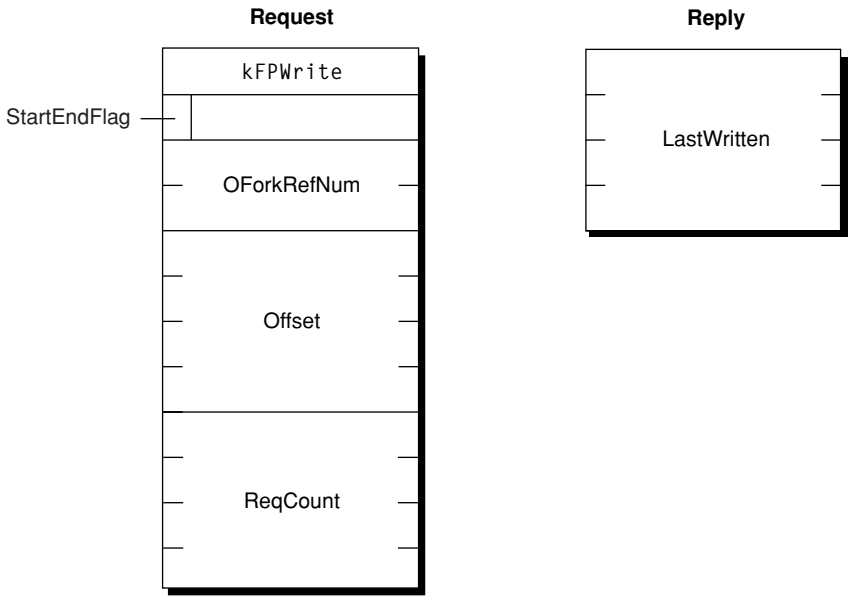
Table 72 lists the result codes for the `FPWrite` command.

Table 72 Result codes for the `FPWrite` command

Result code	Explanation
`kFPAccessDenied`	Fork is not open for writing by this user.
`kFPDiskFull`	No space exists on the volume.
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number or open fork reference number is unknown.

Figure 78 shows the request and reply blocks for the `FPWrite` command.

Figure 78 Request and reply blocks for the `FPWrite` command



FPWriteExt

Writes a block of data to an open fork.

```
byte CommandCode
byte Flag
short OForkRefNum
long long Offset
long long ReqCount
ForkData
```

Parameters

CommandCode

`kFPWriteExt` (61).

Flag

Bit 7 of the `Flag` parameter is the `StartEndFlag` bit, and it indicates whether `Offset` is relative to the beginning or end of the fork. A value of zero indicates that the start is relative to the beginning of the fork; a value of 1 indicates that the start is relative to the end of the fork.

OForkRefNum

Open fork reference number.

Offset

Byte offset from the beginning or the end of the fork indicating where the write is to begin; a negative value indicates a byte within the fork relative to the end of the fork.

ReqCount

Number of bytes to be written.

ForkData

Data to be written, which is not a part of the request block. Instead, the data is transmitted to the server in an intermediate exchange of DSI packets.

Result

`kFPNoErr` if no error occurred. See Table 73 (page 159)for the possible result codes.

ActualCount

Number of bytes actually written to the fork. This long long value is returned by the underlying transport mechanism and is not a value in the reply block.

ReplyBlock

If the result code is `kFPNoErr`, the server returns a reply block consisting of a long, called `LastWritten`, containing the number of the byte just past the last byte written.

Discussion

The server writes data to the open fork, starting at the number of bytes from the beginning or end of the fork as specified by `Offset`.

This command differs from the `FPWrite` (page 155) command in that this command is prepared to handle the large values that may be required for writing to files that reside in volumes larger than 4 GB in size.

The `StartEndFlag` bit indicates whether the block of data is to be written at an offset relative to the beginning or the end of the fork. When the offset is relative to the end of the fork, data can be written without knowing the exact end of the fork, which is useful when multiple writers modify a fork concurrently. The server returns the number of the byte just past the last byte written.

If the block of data to be written extends beyond the end of the fork, the fork is extended. If part of the range is locked by another user, the server returns a `kFPLockErr` result code and does not write any data to the fork.

The file's Modification Date is not changed until the fork is closed.

The fork must be open for writing by the user sending this command.

Lock the range before sending this command. The underlying transport mechanism may force the request to be broken into multiple smaller requests. If the range is not locked when this command begins execution, it is possible for another user to lock some or all of the range before this command completes, causing the write to success partially.

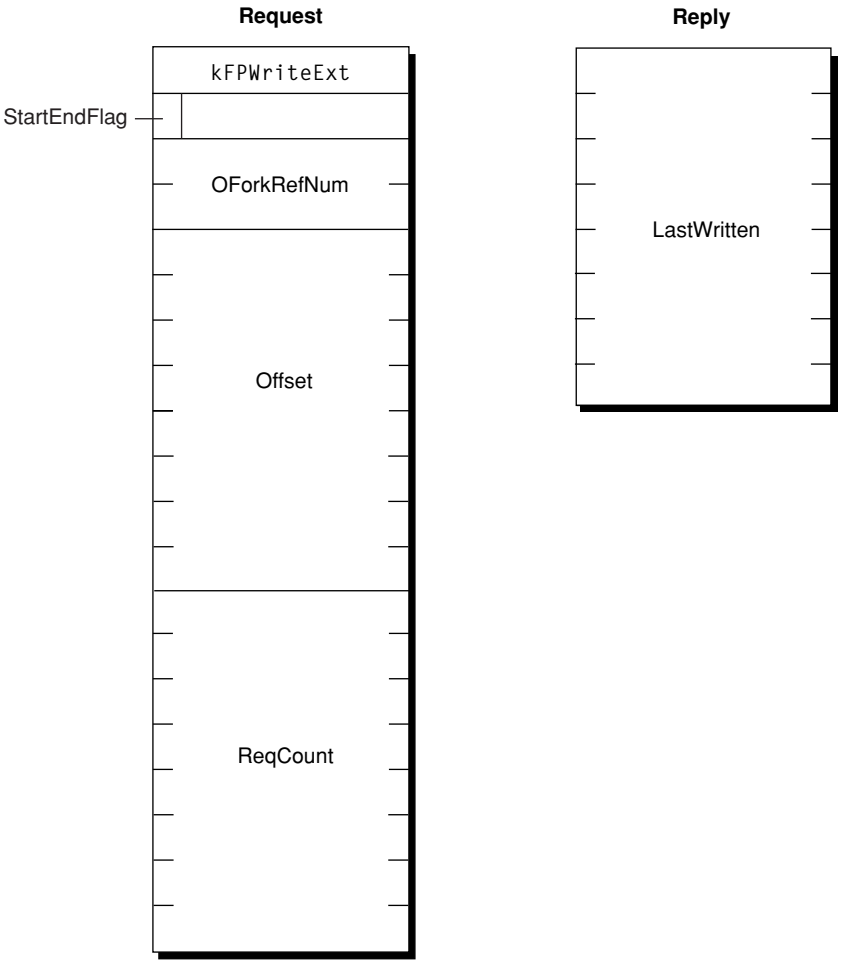
Table 73 lists the result codes for the `FPWriteExt` command.

Table 73 Result codes for the `FPWriteExt` command

Result code	Explanation
`kFPAccessDenied`	Fork is not open for writing by this user.
`kFPDiskFull`	No space exists on the volume.
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number or open fork reference number is unknown.

Figure 79 shows the request and reply blocks for the `FPWriteExt` command.

Figure 79 Request and reply blocks for the `FPWriteExt` command



FPZzzzz

Notifies the server that the client is going to sleep.

```
byte CommandCode
byte Pad
unsigned long Flags
```

Parameters

CommandCode

`kFPZzzzz` (122).

Flag

Reserved.

ReplyBlock

None.

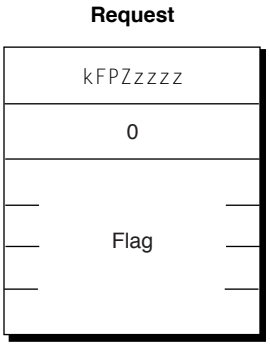
Discussion
If an AFP sharepoint is mounted when the client goes to sleep (for example, an idle sleep or a demand sleep such as when the lid of a PowerBook is closed), the client sends the `FPZzzzz` command. This command notifies the AFP server that the client is going to sleep and that the server should not send any more packets to the client. When the client is awakens, it will send AFP packets to the server, which notifies the server that the client is now awake.

The AFP server should have a setting for the maximum time that a client can sleep — typically 24 hours. If a client has been asleep longer than the maximum sleep time, the server assumes that the client has been disconnected and may free client-related resources on the server.

The `FPZzzzz` command is supported by AFP 2.3 and later over AFP/TCP only.

Figure 80 shows the request block for the `FPZzzzz` command.

Figure 80 Request block for the `FPZzzzz` command



Data Types

Access Control List Structure

Structure that describes a file or directory's access control list (ACL).

```
struct kauth_acl {
    u_int32_t acl_entrycount;
    u_int32_t = acl_flags;
    struct kauth_ace acl_ace[];
};
```

Constants

`acl_entrycount`
Number of `acl_ace` structures.

`acl_flags`
See the Core Foundation ACL documentation for definitions.

`acl_ace`
An `acl_ace` structure. See the Core Foundation ACL documentation for a description of this structure.

Discussion
The Access Control List structure is returned by the `FPGetACL` (page 70) command and set by the `FPSetACL` (page 140) command.

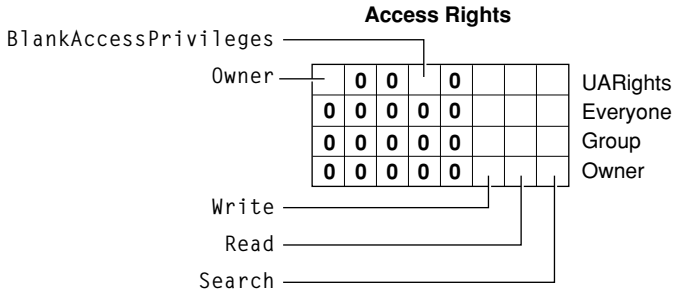
Access Rights Bitmap

A 32-bit value whose bits indicate the ability of the directory's Owner, Group, and Everyone to read, write, and search a directory.

Discussion
Call `FPGetFileDirParms` (page 80) to get the Access Rights bitmap.

Figure 81 (page 162) shows the Access Rights bitmap.

Figure 81 Access Rights bitmap



Directory Bitmap

A 16-bit value whose bits are used to get and set directory parameters.

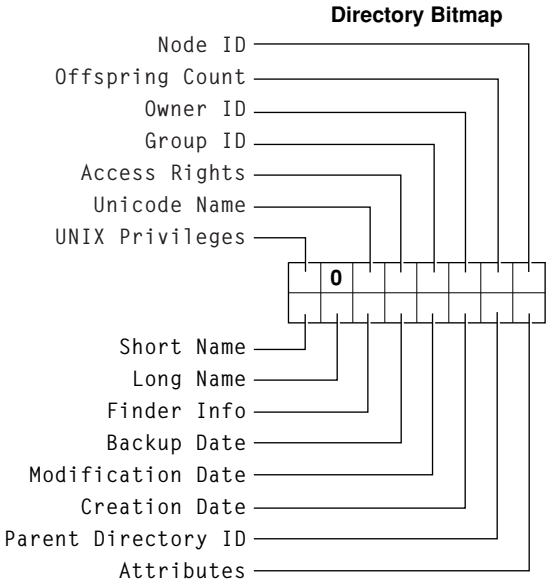
```
enum {
    kFPAttributeBit = 0x1,
    kFPParentDirIDBit = 0x2,
    kFPCreateDateBit = 0x4,
    kFPModDateBit = 0x8,
    kFPBackupDateBit = 0x10,
    kFPFinderInfoBit = 0x20,
    kFPLongNameBit = 0x40,
    kFPShortNameBit = 0x80,
    kFPNodeIDBit = 0x100,
    kFPOffspringCountBit = 0x0200,
    kFPOwnerIDBit = 0x0400,
    kFPGroupIDBit = 0x0800,
    kFPAccessRightsBit = 0x1000,
    kFPProDOSInfoBit = 0x2000 // AFP version 2.2 and earlier
    kFPUTF8NameBit = 0x2000, // AFP version 3.0 and later
    kFPUnixPrivsBit = 0x8000 // AFP version 3.0 and later
};
```

Discussion

The Directory bitmap is used when calling `FPGetFileDirParms` (page 80) to indicate the directory parameters you want to get. It is also used when calling `FPSetDirParms` (page 142) and `FPSetFileDirParms` (page 147) to set directory parameters.

Figure 82 (page 163) describes the Directory bitmap.

Figure 82 Directory bitmap



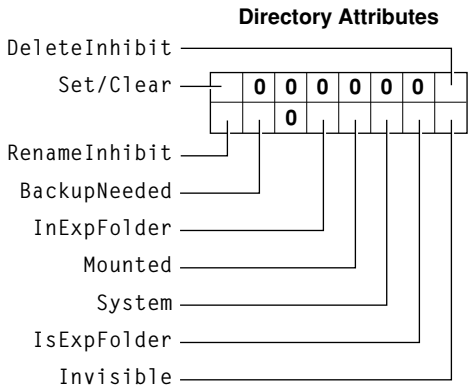
Directory Attributes Bitmap

A 16-bit value whose bits provide additional information about a directory.

Discussion
Use the bits in the Directory Attributes bitmap to inhibit renaming or deleting the directory. Other bits in the Directory Attributes bitmap indicate whether the directory needs to be backed up, whether the directory is mounted by a user, whether the directory is invisible or a system directory, and whether the directory is in a shared area or is a share point. When calling `FPSetDirParms` (page 142) and `FPGetFileDirParms` (page 80) to set Directory Attributes, use the Set/Clear bit (bit 15) to indicate whether you are setting or clearing a directory attribute.

Figure 83 (page 164) describes the Attributes bitmap for a directory.

Figure 83 Directory Attributes bitmap



Extended Attributes Bitmap

Constants that control the behavior when setting extended attributes.

```
enum {
    kXAttrNoFollow = 0x1,
    kXAttrCreate = 0x2,
    kXAttrReplace = 0x4
};
```

Constants
`kXAttrNoFollow`
If set, do not follow symbolic links.

`kXAttrCreate`
If set, `FPSetExtAttr` (page 145) fails if the extended attribute already exists.

`kXAttrReplace`
If set, `FPSetExtAttr` (page 145) fails if the extended attribute does not exist.

Discussion
Use these constants in the Bitmap parameter of the `FPGetExtAttr` (page 77), `FPSetExtAttr` (page 145), and `FPRemoveExtAttr` (page 134) commands to get, set, and remove extended attributes.

File Bitmap

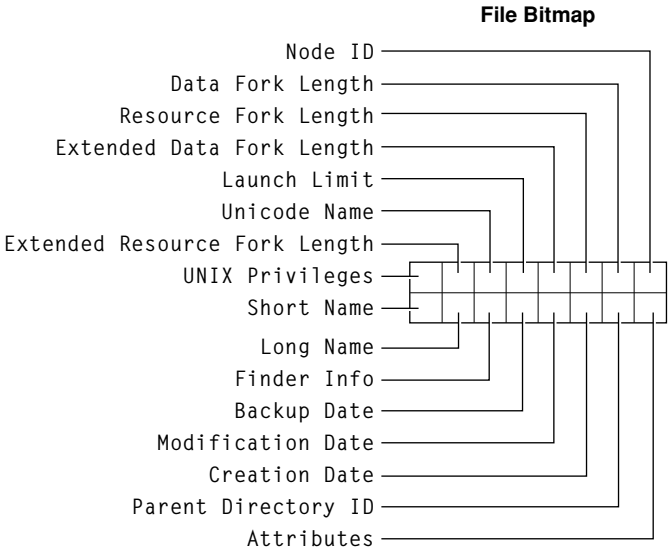
A 16-bit value whose bits are used to get and set file parameters.

```
enum {
    kFPAttributeBit = 0x1,
    kFPParentDirIDBit = 0x2,
    kFPCreateDateBit = 0x4,
    kFPModDateBit = 0x8,
    kFPBackupDateBit = 0x10,
    kFPFinderInfoBit = 0x20,
    kFPLongNameBit = 0x40,
    kFPShortNameBit = 0x80,
    kFPNodeIDBit = 0x100,
    kFPDataForkLenBit = 0x0200,
    kFPRsrcForkLenBit = 0x0400,
    kFPExtDataForkLenBit = 0x0800, // AFP version 3.0 and later
    kFPLaunchLimitBit = 0x1000,
    kFPUTF8NameBit = 0x2000, // AFP version 3.0 and later
    kFPExtRsrcForkLenBit = 0x4000, // AFP version 3.0 and later
    kFPUnixPrivsBit = 0x8000 // AFP version 3.0 and later
};
```

Discussion
The File bitmap is used when calling `FPGetFileDirParms` (page 80) to indicate the file parameters you want to get. It is also used when calling `FPSetFileParms` (page 150) and `FPSetFileDirParms` (page 147) to set file parameters.

Figure 84 (page 165) describes the File bitmap.

Figure 84 File bitmap



File Attributes Bitmap

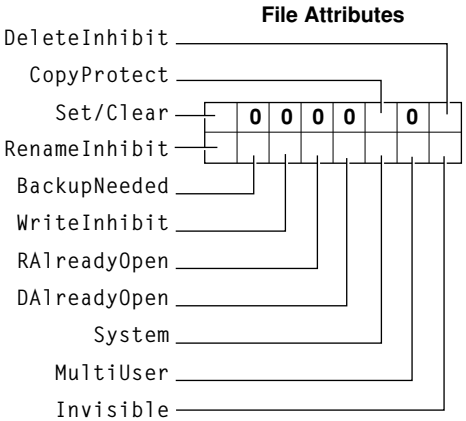
A 16-bit value whose bits provide additional information about a file.

```
enum {
    kFPInvisibleBit = 0x01,
    kFPMultiUserBit = 0x02,
    kFPSystemBit = 0x04,
    kFPDAlreadyOpenBit = 0x08,
    kFPRAlreadyOpenBit = 0x10,
    kFPWriteInhibitBit = 0x20,
    kFPBackUpNeededBit = 0x40,
    kFPRenameInhibitBit = 0x80,
    kFPDeleteInhibitBit = 0x100,
    kFPCopyProtectBit = 0x400,
    kFPSetClearBit = 0x8000
};
```

Discussion
Use the bits in the File Attributes bitmap to inhibit writing, renaming or deleting the file. Other bits in the File Attributes bitmap indicate whether the file needs to be backed up, whether the file can be copied, whether the file is invisible or a system file, whether the file's data or resource fork is open, and whether the file can be opened at the same time by multiple users. When calling `FPSetFileParms` (page 150) and `FPSetFileDirParms` (page 147) to set File Attributes, use the Set/Clear bit (bit 15) to indicate whether you are setting or clearing a file attribute.

Figure 85 (page 166) describes the Attributes bitmap for a file.

Figure 85 File Attributes bitmap



FPUnixPrivs

A structure the describes UNIX privileges for files and directories that reside on a volume that supports UNIX privileges.

```
struct FPUnixPrivs {
    unsigned long uid;
    unsigned long gid;
    unsigned long permissions;
    unsigned long ua_permissions;
};
```

Fields

`uid`

User ID of the file or directory's owner.

`gid`

Group ID of the file or directory's owner.

`permissions`

Setting of the file or directory's permission bits.

`ua_permissions`

User's access rights to the file or directory. Bit 31 is set if the user is the owner of the file or directory.

Discussion

A `FPUnixPrivs` structure is returned when you call `FPGetFileDirParms` (page 80) and specify that you want to get the UNIX privileges for a file or directory.

Server Flags Bitmap

A 16-bit value that describes server capabilities.

```
enum {
    kSupportsCopyfile = 0x01,
    kSupportsChgPwd = 0x02,
    kDontAllowSavePwd = 0x04,
    kSupportsSrvrMsg = 0x08,
    kSrvrSig = 0x10,
    kSupportsTCP = 0x20,
    kSupportsSrvrNotify = 0x40,
    kSupportsReconnect = 0x80,
    kSupportsDirServices = 0x100,
    kSupportsUTF8SrvrName = 0x200,
    kSupportsUUIDs = 0x400,
    kSupportsSuperClient = 0x8000
};
```

Discussion

The Server Flags bitmap is returned by the `FPGetSrvrInfo` (page 91) command.

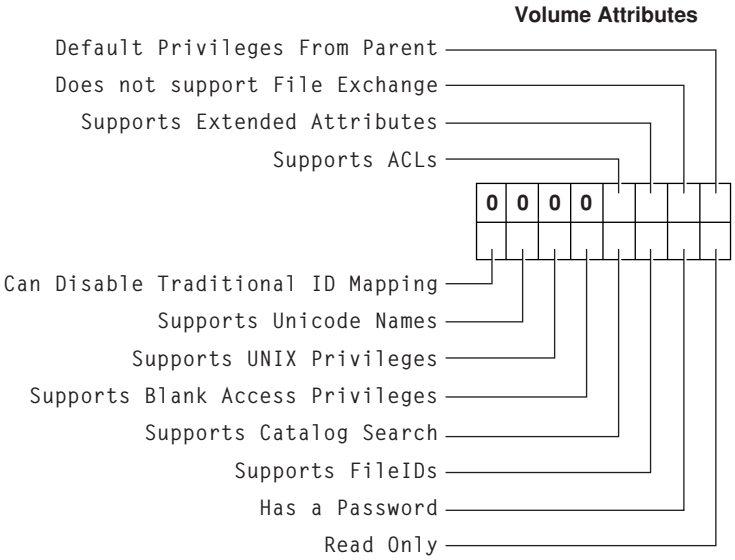
Volume Attributes Bitmap

A 16-bit value whose bits describe how a volume is mounted and whether it supports certain AFP features.

```
enum {
    kReadOnly = 0x01,
    kHasVolumePassword = 0x02,
    kSupportsFileIDs = 0x04,
    kSupportsCatSearch = 0x08,
    kSupportsBlankAccessPrivs = 0x10,
    kSupportsUnixPrivs = 0x20,
    kSupportsUTF8Names = 0x40,
    kNoNetworkUserIDs = 0x80,
    kDefaultPrivsFromParent = 0x10,
    kNoExchangeFiles = 0x20,
    kSupportsExtAttrs = 0x40,
    kSupportsACLs = 0x80
};
```

Discussion
Figure 86 describes the Attributes bitmap for a volume.

Figure 86 Volume Attributes bitmap



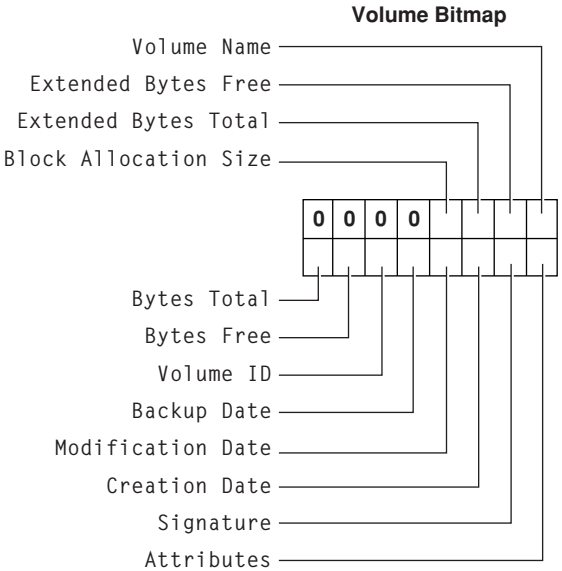
Volume Bitmap

A 16-bit value whose bits are used to get and set volume parameters.

```
enum {
    kFPBadVolPre222Bitmap = 0xFe00,
    kFPBadVolBitmap = 0xF000,
    kFPVolAttributeBit = 0x1,
    kFPVolSignatureBit = 0x2,
    kFPVolCreateDateBit = 0x4,
    kFPVolModDateBit = 0x8,
    kFPVolBackupDateBit = 0x10,
    kFPVolIDBit = 0x20,
    kFPVolBytesFreeBit = 0x40,
    kFPVolBytesTotalBit = 0x80,
    kFPVolNameBit = 0x100,
    kFPVolExtBytesFreeBit = 0x200,
    kFPVolExtBytesTotalBit = 0x400,
    kFPVolBlockSizeBit = 0x800
};
```

Discussion
The Volume bitmap is used when calling `FPGetVolParms` (page 101) to indicate the volume parameters you want to get. It is also used when calling `FPSetVolParms` (page 154) to set a volume's backup date, which is the only Volume parameter that an AFP client can set. Figure 87 (page 169) describes the Volume bitmap.

Figure 87 Volume bitmap



Constants

Access Control List Bitmap

Bitmap for getting and setting access control lists (ACLs).

```
enum {
kFileSec_UUID = 0x01,
kFileSec_GRPUUID = 0x02,
kFileSec_ACL = 0x04,
kFileSec_REMOVEACL = 0x08,
kFileSec_Inherit = 0x10
};
```

Constants

`kFileSec_UUID`
Set this bit to get or set a UUID.

`kFileSec_GRPUUID`
Set this bit to get or set a Group UUID.

`kFileSec_ACL`
Set this bit to get or set an ACL.

`kFileSec_REMOVEACL`
Set this bit toremove an ACL. This bit is not valid when used with the `FPGetACL` (page 70).

`kFileSec_Inherit`
Set this bit any access control entries (ACEs) that have already been inherited. This constant is used only with the `FPSetACL` (page 140) command.

Discussion
Use the Access Control List bitmap with the `FPGetACL` (page 70) and `FPSetACL` (page 140) commands to control the behavior of those commands.

AFP Version Strings

Strings that identify different AFP versions.

```
"AFPVersion 2.1"
"AFP2.2"
"AFP2.3"
"AFPX02"
"AFP3.1"
"AFP3.2"
```

Constants

`AFPVersion 2.1`
AFP version 2.1.

`AFP2.2`
AFP version 2.2.

`AFP2.3`
AFP version 2.3.

`AFPX02`
AFP version 3.0.

`AFP3.1`
AFP version 3.1.

`AFP3.2`
AFP version 3.2.

Discussion
AFP Version strings are returned by the `FPGetSrvrInfo` (page 91) command. AFP clients sent an AFP Version string as a parameter to the `FPLogin` (page 105) and `FPLoginExt` (page 109) commands.

AFP UAM Strings

Strings that identify different UAM versions.

```
"No User Authent"
"Cleartxt Passwrd"
"Randnum Exchange"
"2-Way Randnum"
"DHCAST128"
"DHX2"
"Client Krb v2"
“Recon1”
```

Constants

`No User Authent`
UAM that does not require user authentication.

`Cleartxt Passwrd`
Cleartext Password UAM.

`Randnum Exchange`
Random Number Exchange UAM.

`2-Way Randnum`
Two-Way Random Number Exchange UAM.

`DHCAST128`
Diffie-Hellman Exchange UAM to be used during the log process.

`DHX2`
Causes the Diffie-Hellman Exchange 2 UAM.

`Client Krb v2`
Kerberos UAM.

`Recon1`
Reconnect UAM.

Discussion
AFP UAM strings are returned by the `FPGetSrvrInfo` (page 91) command. AFP clients sent an AFP UAM string as a parameter to the `FPLogin` (page 105) and `FPLoginExt` (page 109) commands.

FPGetSessionToken Types

Values for the `Type` parameter of the `FPGetSessionToken` command.

```
enum {
kLoginWithoutID = 0,
kLoginWithID = 1,
kReconnWithID = 2,
kLoginWithTimeAndID = 3,
kReconnWithTimeAndID = 4,
kRecon1Login = 5,
kRecon1ReconnectLogin = 6,
kRecon1Refresh = 7, kGetKerberosSessionKey = 8
};
```

Constants

`kLoginWithoutID`
The `FPGetSessionToken` parameter block does not contain an `ID` parameter; specified by AFP clients that support a version of AFP prior to AFP 3.1.

`kLoginWithID`
Deprecated.

`kReconnWithID`
Deprecated.

`kLoginWithTimeAndID`
The `FPGetSessionToken` parameter block contains an `ID` and a time stamp parameter. The command is sent to indicate that the client wants its old session to be discarded.

`kReconnWithTimeAndID`
The `FPGetSessionToken` parameter block contains an `ID` and a time stamp parameter. The command is sent to indicate that the client has successfully reconnected and wants the session to be updated with the new value of `ID`.

`kRecon1Login`
Used after logging in to get a credential that can be used to reconnect using the Reconnect UAM. Specifying `kRecon1Login` tells the server to destroy any old sessions that may be associated with the `ID` parameter to the `FPGetSessionToken` command.

`kRecon1ReconnectLogin`
Used to get a new reconnect token after reconnecting using the Reconnect UAM.

`kRecon1RefreshToken`
Used to get a new credential when the current credential is about to expire.

`kGetKerberosSessionKey`
Used to get a Kerberos v5 session key.

Discussion

The value of the `Type` parameter determines the behavior of the server when it receives the`FPGetSessionToken` (page 89) command.

FPMapID Constants

Values used in the `Subfunction` parameter of the `FPMapID` command.

```
enum {
kUserIDToName = 1,
kGroupIDToName = 2,
kUserIDToUTF8Name = 3,
kGroupIDToUTF8Name = 4,
kUserUUIDToUTF8Name = 5,
kGroupUUIDToUTF8Name = 6
};
```

Constants

`kUserIDToName`
Causes `FPMapID` to map the specified User ID to its respective Macintosh Roman user name.

`kGroupIDToName`
Causes `FPMapID` to map the specified Group ID to its respective Macintosh Roman group name.

`kUserIDToUTF8Name`
Causes `FPMapID` to map the specified User ID to its respective user name in UTF-8 encoding.

`kGroupIDToUTF8Name`
Causes `FPMapID` to map the specified Group ID to its respective group name in UTF-8 encoding.

`kUserUUIDToUTF8Name`
Causes `FPMapID` to map the specified User UUID to its respective user name in UTF-8 encoding.

`kGroupUUIDToUTF8Name`
Causes `FPMapID` to map the specified Group UUID to its respective group name in UTF-8 encoding.

Discussion

These constants are used with the `FPMapID` command to indicate the way in which mapping is to occur.

FPMapName Constants

Values used in the `Subfunction` parameter of the `FPMapID` command.

```
enum {
kNameToUserID = 1,
kNameToGroupID = 2,
kUTF8NameToUserID = 3,
kUTF8NameToGroupID = 4,
kUTF8NameToUserUUID = 5,
kUTF8NameToGroupUUID = 6
};
```

Constants

`kNameToUserID`
Causes `FPMapName` to map the specified Macintosh Roman user name to its respective User ID.

`kNameToGroupID`
Causes `FPMapName` to map the specified Macintosh Roman Group name to its respective Group ID.

`kUTF8NameToUserID`
Causes `FPMapName` to map the specified UTF-8-encoded user name to its respective User ID.

`kUTF8NameToGroupID`
Causes `FPMapName` to map the specified UTF-8-encoded group name to its respective Group ID.

`kUTF8NameToUserUUID`
Causes `FPMapName` to map the specified UTF-8-encoded user name to its respective user UUID.

`kUTF8NameToGroupUUID`
Causes `FPMapName` to map the specified UTF-8-encoded user name to its respective Group UUID.

Discussion
These constants are used with the `FPMapName` command to indicate the way in which mapping is to occur.

Path Type Constants

Constants indicating the type of names in a `Pathname` parameter

```
enum {
kFPShortName = 1,
kFPLongName = 2,
kFPUTF8Name = 3
};
```

Constants
`kFPShortName`
Indicates that a `Pathname` parameter contains Short Names.

`kFPLongName`
Indicates that a `Pathname` parameter contains Long Names.

`kFPUTF8Name`
Indicates that a `Pathname` parameter contains an AFPName, which consists of a four-byte text encoding hint followed a two-byte length, followed by a UTF-8– encoded pathname.

Discussion
These constants are used in the `PathType` parameter for many AFP commands to specify the type of names that appear in an associated `Pathname` parameter.

File Creation Constants

Constants used when creating files.

```
enum {
kFPSoftCreate = 0,
kFPHardCreate = 0x80
};
```

Constants
`kFPSoftCreate`
Indicates soft creation.

`kFPLongCreate`
Indicates indicates hard creation.

Discussion
These constants are used in the `Flag` parameter for the `FPCreateFile` (page 45) command.

ACL Access Rights

Access rights bit definitions.

```
#define KAUTH_VNDOE_READ_DATA             (1<<1)
#define KAUTH_VNODE_LIST_DIRECTORY        KAUTH_VNODE_READ_DATA
#define KAUTH_VNODE_WRITE_DATA            (1<<2)
#define KAUTH_VNODE_ADD_FILE              KAUTH_VNODE_WRITE_DATA
#define KAUTH_VNODE_EXECUTE               (1<<3)
#define KAUTH_VNODE_SEARCH                KAUTH_VNODE_EXECUTE
#define KAUTH_VNODE_DELETE                (1<<4)
#define KAUTH_VNODE_APPEND_DATA           (1<<5)
#define KAUTH_VNODE_ADD_SUBDIRECTORY      KAUTH_VNODE_APPEND_DATA
#define KAUTH_VNODE_DELETE_CHILD          (1<<6)
#define KAUTH_VNODE_READ_ATTRIBUTES       (1<<7)
#define KAUTH_VNODE_WRITE_ATTRIBUTES      (1<<8)
#define KAUTH_VNODE_READ_EXTATTRIBUTES    (1<<9)
#define KAUTH_VNODE_WRITE_EXTATTRIBUTES   (1<<10)
#define KAUTH_VNODE_READ_SECURITY         (1<<11)
#define KAUTH_VNODE_WRITE_SECURITY        (1<<12)
#define KAUTH_VNODE_CHANGE_OWNER          (1<<13)
#define KAUTH_VNODE_SYNCHRONIZE           (1<<20)
#define KAUTH_VNODE_GENERIC_ALL           (1<<21)
#define KAUTH_VNODE_GENERIC_EXECUTE       (1<<22)
#define KAUTH_VNODE_GENERIC_WRITE         (1<<23)
#define KAUTH_VNODE_GENERIC_READ          (1<<24)
```

Constants

`KAUTH_VNODE_READ_DATA`
: For a file, the right to read a file's data; for a directory, the right to list the contents of a directory.

`KAUTH_VNODE_LIST_DIRECTORY`
: For a directory, the same as `KAUTH_VNODE_LIST_DIRECTORY`, which is the right to list the contents of a directory.

`KAUTH_VNODE_WRITE_DATA`
: For a file, the right to write to a file; for a directory, the right to create a file in a directory.

`KAUTH_VNODE_ADD_FILE`
: For a file, the same as `KAUTH_VNODE_WRITE_DATA`; the right to write to a file.

`KAUTH_VNODE_EXECUTE`
: Right to execute a program.

`KAUTH_VNODE_SEARCH`
: Same as `KAUTH_VNODE_EXECUTE`.

`KAUTH_VNODE_DELETE`
: Right to delete a file.

`KAUTH_VNODE_APPEND_DATA`
: For a file, the right to append data to a file; for a directory, the right to create a subdirectory in a directory.

`KAUTH_VNODE_ADD_SUBDIRECTORY`
: For a directory, the same as `kAUTH_VNODE_APPEND_DATA`, which is the right to create a subdirectory in a directory.

`KAUTH_VNODE_DELETE_CHILD`
: Right to delete a directory and all the files it contains.

`KAUTH_VNODE_READ_ATTRIBUTES`
: Right to read a file's hidden attributes, such as hidden, read-only, system, and archive.

`KAUTH_VNODE_WRITE_ATTRIBUTES`
: Right to write a file's attributes, such as hidden, read-only, system, and archive.

`KAUTH_VNODE_READ_EXTATTRIBUTES`
: Right to read a file or directory's extended attributes.

`KAUTH_VNODE_WRITE_EXTATTRIBUTES`
: Right to write extended attributes.

`KAUTH_VNODE_READ_SECURITY`
: Right to get a file or directory's access rights.

`KAUTH_VNODE_WRITE_SECURITY`
: Right to set a file or directory's access rights.

`KAUTH_VNODE_CHANGE_OWNER`
: Right to change the owner of a file or directory.

`KAUTH_VNODE_SYNCHRONIZE`
: Right to block until the file or directory is put in the signaled state; provided for Windows interoperability.

`KAUTH_VNODE_GENERIC_ALL`
: Windows NT right that includes all rights specified by `KAUTH_VNODE_GENERIC_EXECUTE`, `KAUTH_VNODE_GENERIC_WRITE`, and `KAUTH_VNODE_GENERIC_READ`.

`KAUTH_VNODE_GENERIC_EXECUTE`
: Windows NT right that in Windows 2000 became the right to read attributes, read permissions, traverse folders, and execute files.

`KAUTH_VNODE_GENERIC_WRITE`
: Windows NT right that in Windows 2000 became right to read access rights, create a subdirectory in a directory, write data in a file, create files in a directory, append data to a file, write attributes, and write extended attributes.

`KAUTH_VNODE_GENERIC_READ`
: Windows NT right that in Windows 2000 became right to list directories, read file data, read attributes, read extended attributes, and read access rights.

Discussion
These definitions are made in `kauth.h`. Use these definitions to specify access rights when calling `FPAccess` (page 11), to parse the access rights returned by `FPGetACL` (page 70), and to set access rights when calling `FPSetExtAttr` (page 145).

Result Codes

The result codes specific to the Apple Filing Protocol are listed in the table below.

Result Code	Value	Description
`kASPSessClosed`	-1072	ASP session closed.
`kFPAccessDenied`	-5000	User does not have the access privileges required to use the command.
`kFPAuthContinue`	-5001	Authentication is not yet complete.
`kFPBadUAM`	-5002	Specified UAM is unknown
`kFPBadVersNum`	-5003	Server does not support the specified AFP version.

Result Code	Value	Description
kFPBitmapErr	-5004	Attempt was made to get or set a parameter that cannot be obtained or set with this command, or a required bitmap is null
kFPCantMove	-5005	Attempt was made to move a directory into one of its descendent directories.
kFPDenyConflict	-5006	Specified fork cannot be opened because of a deny modes conflict.
kFPDirNotEmpty	-5007	Directory is not empty.
kFPDiskFull	-5008	No more space exists on the volume
kFPEOFErr	-5009	No more matches or end of fork reached.
kFPFileBusy	-5010	When attempting a hard create, the file already exists and is open.
kFPFlatVol	-5011	Volume is flat and does not support directories.
kFPItemNotFound	-5012	Specified APPL mapping, comment, or icon was not found in the Desktop database; specified ID is unknown.
kFPLockErr	-5013	Some or all of the requested range is locked by another user; a lock range conflict exists.
kFPMiscErr	-5014	Non-AFP error occurred.
kFPNoMoreLocks	-5015	Server's maximum lock count has been reached.
kFPNoServer	-5016	Server is not responding.
kFPObjectExists	-5017	File or directory already exists.
kFPObjectNotFound	-5018	Input parameters do not point to an existing directory, file, or volume.
kFPParamErr	-5019	Session reference number, Desktop database reference number, open fork reference number, Volume ID, Directory ID, File ID, Group ID, or subfunction is unknown; byte range starts before byte zero; pathname is invalid; pathname type is unknown; user name is null, exceeds the UAM's user name length limit, or does not exist, MaxReplySize is too small to hold a single offspring structure, ThisUser bit is not set, authentication failed for an undisclosed reason, specified user is unknown or the account has been disabled due to too many login attempts; ReqCount or Offset is negative; NewLineMask is invalid.
kFPRangeNotLocked	-5020	Attempt to unlock a range that is locked by another user or that is not locked at all.
kFPRangeOverlap	-5021	User tried to lock some or all of a range that the user has already locked.
kFPSessClosed	-5022	Session is closed.
kFPUserNotAuth	-5023	UAM failed (the specified old password doesn't match); no user is logged in yet for the specified session; authentication failed; password is incorrect.

Result Code	Value	Description
kFPCallNotSupported	-5024	Server does not support this command.
kFPObjectTypeErr	-5025	Input parameters point to the wrong type of object.
kFPTooManyFilesOpen	-5026	Server cannot open another fork.
kFPServerGoingDown	-5027	Server is shutting down.
kFPCantRename	-5028	Attempt was made to rename a volume or root directory.
kFPDirNotFound	-5029	Input parameters do not point to an existing directory.
kFPIconTypeError	-5030	New icon's size is different from the size of the existing icon.
kFPVolLocked	-5031	Volume is Read Only.
kFPObjectLocked	-5032	File or directory is marked DeleteInhibit; directory being moved, renamed, or moved and renamed is marked RenameInhibit; file being moved and renamed is marked RenameInhibit; attempt was made to open a file for writing that is marked WriteInhibit; attempt was made to rename a file or directory that is marked RenameInhibit.
kFPContainsSharedErr	-5033	Directory contains a share point.
kFPIDNotFound	-5034	File ID was not found. (No file thread exists.)
kFPIDExists	-5035	File already has a File ID.
kFPDiffVolErr	-5036	Wrong volume.
kFPCatalogChanged	-5037	Catalog has changed.
kFPSameObjectErr	-5038	Two objects that should be different are the same object.
kFPBadIDErr	-5039	File ID is not valid.
kFPPwdSameErr	-5040	User attempted to change his or her password to the same password that is currently set.
kFPPwdTooShortErr	-5041	User password is shorter than the server's minimum password length, or user attempted to change password to a password that is shorter than the server's minimum password length.
kFPPwdExpiredErr	-5042	User's password has expired.
kFPInsideSharedErr	-5043	Directory being moved contains a share point and is being moved into a directory that is shared or is the descendent of a directory that is shared.
kFPInsideTrashErr	-5044	Shared directory is being moved into the Trash; a directory is being moved to the trash and it contains a shared folder.
kFPPwdNeedsChangeErr	-5045	User's password needs to be changed.

Result Code	Value	Description
kFPPwdPolicyErr	-5046	New password does not conform to the server's password policy.
kFPDiskQuotaExceeded	-5047	Disk quota exceeded.

Document Revision History

This table describes the changes to *Apple Filing Protocol Reference*.

Date	Notes
2006-05-23	Updated the result code table.
2006-04-04	First publication of this content as a separate document.

Document Revision History

Index

K

N

P

R

S

V