AppleShare X
Developer's Kit

Apple Filing Protocol Version 3.0

Technical Publications

April 10, 2000. Draft. Preliminary.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

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

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

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

Apple, the Apple logo, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and / or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

QuickView™ is licensed from Altura Software, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

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

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

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

Contents

Preface About This Manual vii

Chapter 1 About Apple Filing Protocol Version 3.0 11

Figures, Tables, and Listings

Chapter 1 About Apple Filing Protocol Version 3.0 11

Chapter 2 Apple Filing Protocol Reference 39

About This Manual

This document describes the version 2.1, 2.2, and 3.0 extensions to version 2.0 of the Apple Filing Protocol (AFP), which is documented in *Inside AppleTalk*. The AFP 2.1 extensions support extra features in AFP servers and new calls that were added to the hierarchical file system (HFS) for System 7. The AFP 2.2 extensions support new features introduced in AppleShare IP 5.0. The AFP 3.0 extensions support new features introduced in AppleShare X.

Conventions Used in This Manual

The Courier font is used to indicate server control calls, code, and text that you type. Terms that are defined in the glossary appear in boldface at first mention in the text. This guide includes special text elements to highlight important or supplemental information:

Note
Text set off in this manner presents sidelights or interesting points of information. ◆

IMPORTANT
Text set off in this manner—with the word Important—presents important information or instructions. ▲

▲ **WARNING**
Text set off in this manner—with the word Warning—indicates potentially serious problems. ▲

For more information

The following books provide information that is important for all AppleShare developers:

- *AppleShare IP Administrator's Manual*. Apple Computer, Inc.
- *Inside Macintosh*. Apple Computer, Inc.

For information about the programming interface for managing users and groups, see the following publication:

- *AppleShare X Developer's Kit*: *AppleShare Registry Library*. Apple Computer, Inc.

For additional information on the Apple Filing Protocol (AFP), see the following publications:

- *AppleShare X Developer's Kit*: *Apple Filing Protocol*. Apple Computer, Inc.
- *Inside AppleTalk*, Second Edition. Apple Computer, Inc.

For information on user authentication modules (UAMs), see the following publication:

- *AppleShare X Developer's Kit*: *User Authentication Modules*. Apple Computer, Inc.

For information on controlling an AppleShare file server and handling server events, see the following publication:

- *AppleShare X Developer's Kit*: *Server Control Calls and Server Event Handlng.* Apple Computer, Inc.

For information on AppleShare IP Print Server security mechanisms, see the following publication:

- *AppleShare X Developer's Kit*: *AppleShare IP Print Server Security Protocol*. Apple Computer, Inc.

For information on using the AppleShare X File Server and Macintosh File Sharing, see the following manuals:

- *AppleShare Client User’s Manual.* Apple Computer, Inc.
- *Macintosh Networking Reference*. Apple Computer, Inc.

About Apple Filing Protocol Version 3.0

This document describes modifications to the Apple Filing Protocol (AFP) for version 2.1, 2.2, and 3.0. The changes are summarized as follows:

- AFP 2.1 support extra features in AFP servers and new calls that were added to the hierarchical file system (HFS) for System 7.
- AFP 2.2 support new features introduced in AppleShare IP 5.0.
- AFP 3.0 supports new features introduced in AppleShare X.

AFP version 2.0 is documented in *Inside AppleTalk*.

Table 1-1 lists all of the AFP version strings.

Table 1-1 AFP version strings

AFP version	AFP version string
AFP 1.1	AFPVersion 1.1
AFP 2.0	AFPVersion 2.0
AFP 2.1	AFPVersion 2.1
AFP 2.2	AFP2.2
AFP3.0	AFP3.0

Note
AFP version 1.0 was not released. ◆

About AFP Version 3.0

This section describes changes to AFP since AFP version 2.2.

Longer Pathnames

AFP 3.0 supports a new path type, `kFPUTF8Name`, which begins with a two-byte length field and is followed by a data field. The length field specifies the length of valid data in the data field. The two-byte length field allows AFP 3.0 to support longer pathnames and pathnames that consists of Unicode characters.

All calls that take an AFP pathname as a parameter can take a pathname of type `kFPUTF8Name`.

Changes to the File and Directory Bitmaps

AFP 3.0 redefines bits in the file and directory bitmaps. Prior to AFP 3.0, the following constant was defined:

```
kFPProDOSInfoBit = 0x2000
```

In AFP 3.0, the following constant is defined for the same bits:

```
kFPUTF8NameBit = 0x2000
```

In AFP 3.0, the following constants are defined for the file bit map:

```
    kFPExtDataForkLenBit= 0x0800
    kFPExtRsrcForkLenBit= 0x4000
    kFPUnixPrivsBit     = 0x8000 (Reviewers: Not certain if this is in
this version)
```

New Commands

AFP 3.0 supports several new commands:

- `FPByteRangeLockExt`, an extended range lock command

- `FPGetAuthMethods`, a command for getting the authentication methods the server supports
- `FPLongExt`, a command that logs on to an AFP server using the specifed login directory.
- `FPReadExt`, an extended read command
- `FPWriteExt`, an extended write command

Extended Subfunction Codes for FPMapID and FPMapName

For AFP 3.0, the subfunction codes for the `FPMapID` and `FPMapName` commands have been extended. Table 1-2 lists all of the subfunction codes for the FPMapID command.

Table 1-2 Subfunction codes for FPMapID

Subfunction code	Purpose
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 UTF8 user name
4	Maps a group ID to a UTF8 group name

Table 1-3 lists all of the subfunction cods for the `FPMapName` command.

Table 1-3 Subfunction codes for FPMapName

Subfunction code	Purpose
1	Maps a Macintosh Roman user name to a user ID
2	Maps a Macintosh Roman group name to a group ID
3	Maps a UTF8 user name to a user ID
4	Maps a UTF8 group name to a group ID

Changes to Attention Messages

For AFP 3.0, an AFP client can inform the server that it can accept large attention messages (for example, .5K to 3K). If the server chooses, it may then include the attention message in the initial attention request instead of in a later packet.

About AFP Version 2.2

The following commands were modified for AFP version 2.2:

- `FPGetSrvrInfo` (page 61), which retrieves information about the server, including its name, machine type, the AFP versions and user authentication methods it supports, the server's unique identifier, and its AFP network address.
- `FPGetVolParms` (page 69), which retrieves information about a particular volume, such as the creation date, modification date, backup date, total size, number of free bytes, and volume name.
- `FPOpenVOL`, which with AFP 2.2 uses the same bitmap as `FPGetVolParms` (page 69).

In addition, AFP version 2.2 implements server notifications as an attention code. For details, see Table 1-8 (page 27).

About AFP Version 2.1

The following commands were added to AFP version 2.1:

- `FPGetSrvrMsg` (page 66), which enables an AFP client to get a string message from the server. Support for this command is optional; a server can be considered AFP 2.1–compliant regardless of whether it supports this command.
- `FPCreateID` (page 51), `FPDeleteID` (page 53), `FPResolveID` (page 77), and `FPExchangeFiles` (page 55), which support file IDs. File IDs provide a mechanism by which applications and users can keep track of a file even if it has been moved or its name has been changed. Support for these commands

is optional. For more information, see “Bitmap for FPGetVolParms” (page 23).

- FPCatSearch (page 44), which allows searching of the catalog on almost any field that is returned by PBGetCatInfo. Support for this command is optional. For more information, see “Bitmap for FPGetVolParms” (page 23).

AFP 2.1 also defines changes in the behavior of the server to support optional enhanced security features.

To accommodate some of the new features of AFP version 2.1 and HFS, the bitmaps of certain commands were modified:

- new Directory Attributes and Access Rights returned by FPGetFileDirParms and any command that uses this bitmap
- new bit definitions in the Flags word returned by FPGetSrvrInfo
- new Volume Attributes returned by FPGetVolParms

A user authentication method (UAM) known as Two-Way Random Number Exchange was introduced with AFP 2.1. When this method is used, not only is the user authenticated to the server, but the server is authenticated to the user.

A “blank access privileges” feature was added. It is designed to be used on a local computer in which some portions of the hierarchical file system are shared (or “exported”) for regular users, while the entire hierarchy is available for the local user (and the owner when connected remotely). A folder with blank access privileges “inherits” the privileges of the folder in which it is contained.

Furthermore, when a folder is created remotely, the default access privileges assigned to that folder are different under AFP 2.1 than under AFP 2.0. When a user creates a new folder under AFP 2.1, the owner is still assigned full privileges, but the enclosing folder’s Group and Everyone privileges are copied to the new folder.

In AFP 2.1, user and group names are valid in either the owner field or the group field. This enhancement allows for two new situations that were not allowed under AFP 2.0:

- A folder can now be owned by more than one user.
- A different set of access privileges for a shared folder can be assigned for a user (or group) than for everyone else.

Blank Access Privileges

AFP version 2.1 and later supports blank access privileges for folders. When a folder's blank access privileges bit is set, then its other access privilege bits are ignored and it uses the access privilege bits of its parent. The inherited access privileges include the parent's group affiliation.

Blank access privileges cannot be set on any share point. Since the volume root directory (directory ID = 2) of a shared volume is always a share point for the administrator / owner, blank access privileges cannot be set on a volume root directory.

IMPORTANT

This paradigm is useful because it causes folders' access privileges to behave as users expect them to: When a folder with blank access privileges is moved around within a folder hierarchy, it always reflects the access privileges of the folder containing it. However, when the blank-access-privileges bit is cleared for a folder, its current access privileges "stick" to that folder and remain unchanged no matter where the folder is moved. Therefore, although the use of blank access privileges is optional under AFP 2.1, it is highly recommended that you include this feature in your AFP 2.1 implementation as it has subtle human interface repercussions. ▲

Two-Way Random Number Exchange UAM

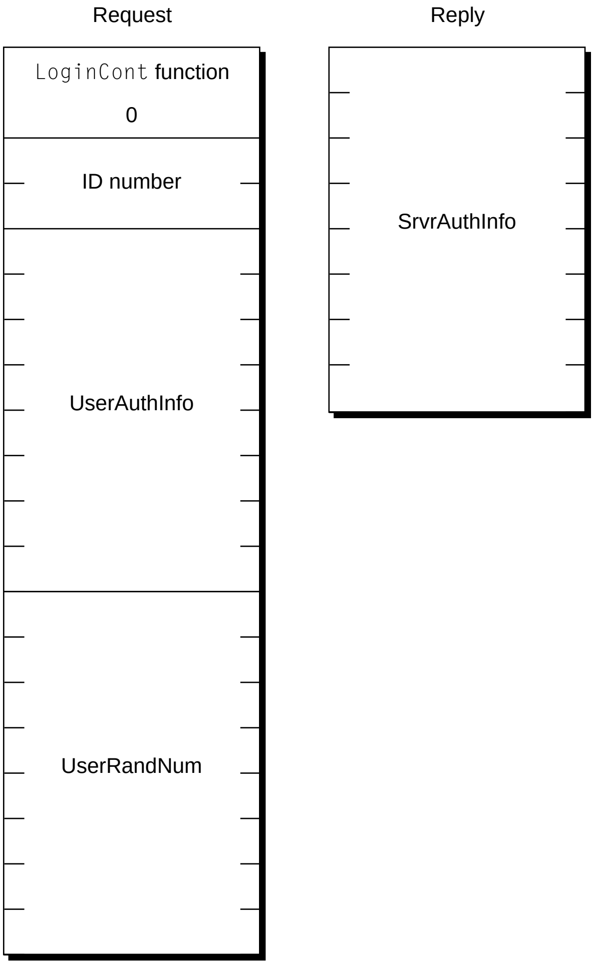
AFP version 2.1 and later supports a user authentication method known as the Two-Way Random Number Exchange UAM. With this UAM, the user is authenticated to the server and the server is also authenticated to the user. This method uses the same initial steps as the Random Number Exchange UAM, with one additional step. The corresponding UAM string is 2-Way Randnum exchange.

Both the Random Number Exchange UAM and the Two-Way Random Number Exchange UAM start with the client asking to log on to the server. If the logon is allowed, the server returns a 2-byte ID number, an 8-byte random number challenge and an error of `afpAuthContinue`. The client then encodes the challenge with its password and sends the encoded challenge along with the ID number back to the server in an `FPLoginCont` command block. If the encoded password is correct, the client is authenticated and `noErr` is returned. However for the Two-Way Random Number Exchange UAM, the client sends a second 8-byte random number challenge, the server encodes the client challenge with what it believes is the user’s password and returns the encoded challenge in the `FPLoginCont` reply.

The client compares this response with what resulted from its encoding of the client challenge; if the two are identical, the server is also authenticated. This feature gaurds against spoofing (that is, using a fake server to get passwords or data).

Figure 1-1 shows the request and reply block formats for the `FPLoginCont` command when the Two-Way Random Number Exchange UAM is used.

Figure 1-1 Request and reply blocks for Two-Way Random Number Exchange



The Two-Way Random Number Exchange UAM is not available for use with the `FPChangePassword` command, nor is it required. If the user is concerned about authenticating the server, he or she will have already logged on to the server with the Two-Way Random Number Exchange UAM. Since the user must already be authenticated to call `FPChangePassword`, he or she is assured that the server is the one expected.

UAM Implementation Notes

Both the Random Number Exchange UAM and the Two-Way Random Number Exchange UAM use 8-bit ASCII characters in the password. Seven-bit ASCII is used only by the Cleartext UAM.

The Random Number Exchange and Two-Way Random Number Exchange UAMs interpret differently the password used as the key passed to the **National Institute of Standards and Technology** Data Encryption Standard (NIST DES) algorithm (The NIST is formerly known as the National Bureau of Standards [NBS].)

With the Random Number Exchange UAM, the key (password) is used without change. Thus, the low-order bit of each byte of the password is ignored. The NIST DES algorithm uses only 56 bits of the 64-bit key, and the unused bits are where the low-order bit of each password character is kept. The result is that in passwords, “0” matches “1”, “b” matches “c”, and so on.

With the Two-Way Random Number Exchange UAM, the key is shifted left 1 bit before it is used, so that the high-order bit is ignored. Two values are still accepted for each byte of the password. However, the two values will not be adjacent in ASCII space and so will probably not be adjacent alphabetically. (For example, “0” will match “∞”, “7” will match “∑”, and so on.)

Modified Bitmap Definitions

This section describes the bitmaps defined for AFP 2.1 and later. The bitmap definitions are divided into three categories:

- the Directory Attributes and Access Privileges words for the `FPGetFileDirParms` command
- the Flags word for the `FPGetSrvrInfo` command
- the Volume Attributes word for the `FPGetVolParms` command.

Bitmaps for FPGetFileDirParms

To accommodate the ability to share folders within Macintosh File Sharing and AppleShare 3.0 (as opposed to the ability to share only entire volumes under

AppleShare 2.0.1), the bit definitions shown in Table 1-4 were added to the Directory Attributes word for FPGetFileDirParms for AFP version 2.1 and later.

Table 1-4 Bit definitions added to the Directory Attributes word for AFP 2.1 and later

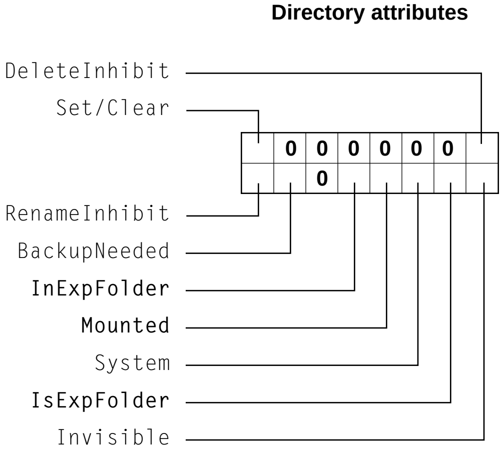
Bit	Meaning
IsExpFolder (bit 1)	This folder is a share point. This folder, and all folders within it, will give feedback to the local user, indicating that access privileges are valid (for example, by using tabbed folders or drop-box folder icons, or by enabling the Get Privileges [System 6] or Sharing [System 7] menu items). None of the folders outside the shared (exported) area show access privileges on the local computers (although they may still possess valid access privilege information, which only an administrator can see or modify).
Mounted (bit 3)	This share point is mounted by a user who is not an administrator. The icon for such a folder indicates to the user of the local computer that this folder is a share point, and that a remote user currently has it mounted.
InExpFolder (bit 4)	This folder is in a shared area of the folder hierarchy. This folder, and all folders within it, will give feedback to the local user, indicating that access privileges are valid. This folder cannot be shared, since a share point cannot exist within another share point.

Note

IsExpFolder, Mounted, and InExpFolder are read-only; they cannot be set with FPSetFileDirParms. They are returned to the remote user and are relevant to a general AFP server. The reason is that the administrator / owner can access the whole server from the volume root directory down, and regular users can access only those portions of the volume that are contained within the share points (which may be contained within the volume directory level). ◆

Figure 1-2 shows the entire Directory Attributes word, with the added bits shown in boldface.

Figure 1-2 Directory Attributes word



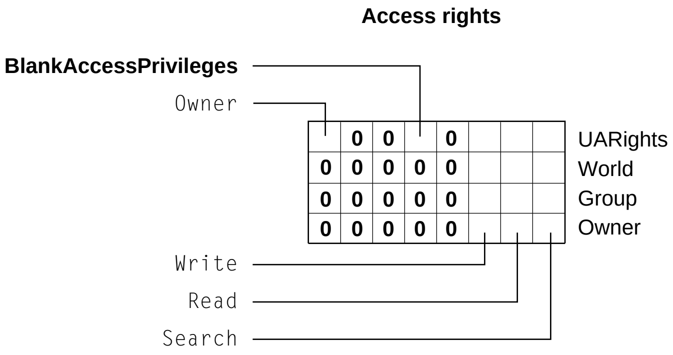
To accommodate blank access privileges, the bit definition shown in Table 1-5 was added to the Access Rights long word for the `FPGetFileDirParms` for AFP version 2.1 and later.

Table 1-5 Bit definition added to the Access Rights long word for AFP 2.1 and later

Bit	Meaning
`BlankAccessPrivileges` (bit 28)	This folder has blank access privileges and will have the same access privileges as the folder enclosing it.

Figure 1-3 shows the entire Access Rights long word, with the added bit shown in boldface.

Figure 1-3 Access Rights long word



Bitmap for FPGetSrvrInfo

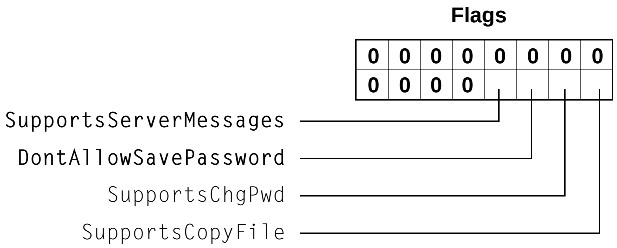
To accommodate optional new features in AFP 2.1, bit definitions shown in Table 1-6 were added to the Flags word for `FPGetSrvrInfo`.

Table 1-6 Bit definitions added to the Flags word for FPGetSrvrInfo for AFP 2.1 and later

Bit	**Meaning**
`DontAllowSavePassword` (bit 2)	The client should not allow the user to save his or her password for volumes mounted at system startup. The item-selection dialog box may still allow the user to save his or her name. However, when this bit is set, the button offering that option is not displayed.
`SupportsServerMessages` (bit 3)	Since server messages are an option in AFP 2.1, this bit allows servers to specify whether this optional feature is supported.

Figure 1-4 shows the entire Flags word, with the added bits shown in boldface.

Figure 1-4 Flags word



For information about additional changes made to the `FPGetSrvrInfo` bitmap for AFP 2.2, see `FPGetSrvrInfo` (page 61).

Bitmap for FPGetVolParms

To accommodate new HFS functionality in System 7, the bit definitions shown in Table 1-7 were added to the Volume Attributes word for `FPGetVolParms`.

Table 1-7 Bit definitions added to the Volume Attributes word for AFP 2.1 and later

Bit	Meaning
`HasVolumePassword` (bit 1)	This volume has a volume password. Volume passwords were supported in prior versions of AFP; now the volume attributes reflect this information. This bit has the same value as the `HasPassword` bit returned for each volume by `FPGetSrvrParms`.
`SupportsFileIDs` (bit 2)	This volume supports file IDs. In general, if file IDs are supported on one volume, they will be supported on all volumes, but this bit allows the server to be more selective, if necessary.

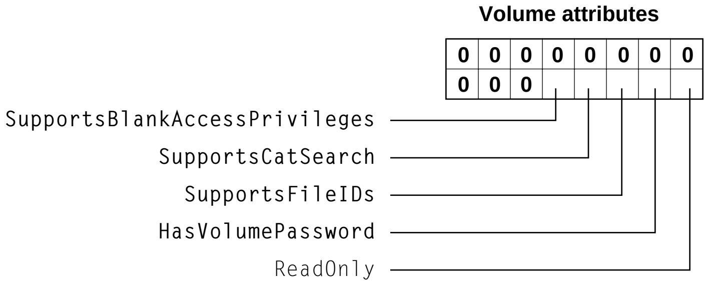
continued

Table 1-7 Bit definitions added to the Volume Attributes word for AFP 2.1 and later (continued)

Bit	Meaning
SupportsCatSearch (bit 3)	This volume supports the FPCatSearch command. Since the use of FPCatSearch is optional in AFP 2.1, this bit allows the server to make this capability available on a per-volume basis.
SupportsBlankAccessPrivileges (bit 4)	This volume supports blank (inherited) access privileges.

Figure 1-5 shows the entire Volume Attributes word, with the new bits for AFP version 2.1 in boldface.

Figure 1-5 Volume Attributes word



Security Features

This section describes the security features of AFP version 2.1 and later: minimum password length, password expiration, and maximum failed logon attempts.

Minimum Password Length

With AFP version 2.1 and later, you can specify the minimum length for a user's password. This length is specified by means of some administrative program. If the user's password is too short, he or she will get an afpPwdTooShortErr error upon logging on. The client code should display an explanatory dialog box and then allow the user to change his or her password. The FPChangePassword command will continue to fail with an afpPwdTooShortErr error until a password of at least the specified length is submitted.

The administrative program should be intelligent enough to prevent the administrator from giving users passwords that are too short; otherwise these users' first logon attempts will be dissatisfying, if not confusing. Whether or not the administrative program should alert the administrator when passwords for existing users are too short (as might happen when the administrator changes the minimum password length from 4 to 8) is up to the developer of the administrative program. The maximum password length is still 8.

Password Expiration

With AFP version 2.1 and later, you can specify the period of time after which a user must change his or her password. This interval can be specified by means of a server administrative program. If the user changes the password before the password expiration time expires, the password expiration timer is reset. If the user does not change the password before the interval expires, the actions that he or she can perform become severely limited. If the workstation is using AFP 2.1, the user can issue an FPChangePassword command and change the password, issue an FPLogout command, or issue an FPLoginCont command. (If the user issues any other command, the error FPParmErr is returned.) The FPLoginCont command returns one of the following errors: afpPwdTooShortErr, afpPwdExpiredErr, or afpPwdNeedsChangeErr. At this point the user is logged on, and the only command that can be issued is FPChangePassword or FPLogout. If the user issues any other command, the error FPParmErr is returned. Once the user successfully changes the password, the user can issue any command.

Note that if the workstation is using a version of AFP earlier than 2.1, two additional calls, FPGetSrvrParms and FPOpenVol, allow the user to log on as usual without returning an error.

If the administrator wants to give a user an account that becomes inactive after a certain interval, the administrator can set the password expiration time to that interval and then disallow the changing of the password. When the time expires, the user is no longer able to connect to the server.

To keep users from circumventing this feature, a new error, afpPwdSameErr, is returned by the FPChangePassword command. This error prevents the user from changing his or her password when the new password is the same as the old password. The FPChangePassword command returns afpPwdSameErr only if the password expiration feature is enabled.

Maximum Failed Logon Attempts

With AFP version 2.1 and later, you can specify the maximum number of consecutive failed logon attempts that will be allowed before the user's account is disabled. This count can be specified by an administrative program. The count is reset to zero after every successful logon. For every failed logon attempt without a preceding successful logon, the count is incremented. When the maximum number of failed logon attempts is reached, the user's account is disabled. Any attempts to log on after the account is disabled result in an FPParmErr 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, such as a phone call.

Changes to AFPUserBytes Definitions

The AFPUserBytes bytes make up the 2-byte attention code sent in an ASP Attention packet to the AFP client. This section describes how the AFPUserBytes bytes were augmented to accommodate AFP 2.1 features (such as the server message feature) and modes in the workstation code (such as Disconnect) and new capabilities in the client code (such as auto-reconnect). For AFP 2.2, the attention code 0011 represents a server notification (see Table 1-8).

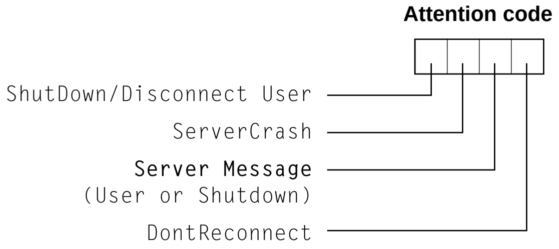
The AFPUserBytes bytes are shown in Figure 1-6.

Figure 1-6 AFPUserBytes

Attention code (4 bits)	Number of minutes or extended bitmap (12 bits)

Figure 1-7 shows how the attention code bits for the AFPUserBytes bytes are defined, with the bit definitions for AFP 2.1 in boldface.

Figure 1-7 Attention code bits in AFPUserBytes



The bit numbers for the attention code bits are defined in Table 1-8.

Table 1-8 Attention code bits

Bit	Meaning
15	Shutdown or Attention bit. This bit is used when the server is being shut down or one or more users are being disconnected.
14	Server Crash bit. The server has detected an internal error, and the session will close immediately with minimal flushing of files. There may be some data loss. This condition is never accompanied by a server message and is highly unlikely to occur.
13	Server Message bit. There is a server message that the client should request by calling `FPGetSrvrMsg` with a MsgType of “Server.” For more information, see the `FPGetSrvrMsg` (page 66). The client should request the message as soon as possible after receiving this attention code. Otherwise, the server message it receives could be out of date.
12	Don’t Reconnect bit. This bit is set when the user is disconnected, so that the client’s reconnect code does not attempt to reconnect the session. This bit is not set for normal server shutdowns and is not set when the server loses power or when there is a break in network cabling. This mechanism allows administrators to shut down the server for backup purposes, bring the server up, and allow disconnected clients to reconnect transparently. This bit is ignored when the number of minutes is any value other than zero.

continued

Table 1-9 lists valid combinations for the attention code bits.

Table 1-9 Valid combinations for the Attention Code bits

Combination	Meaning
1000	The server is shutting down in the designated number of minutes, or the user will be disconnected in the designated number of minutes. No message accompanies this shutdown. This attention code may be used when the server shuts down (that is, when the administrator quits file service).
1001	The server is shutting down, or the user will be disconnected in the designated number of minutes. No message accompanies this shutdown. This attention code is used upon user disconnection (for example, when the administrator detects an intruder and disconnects him or her).
1010	The server is shutting down, or the user will be disconnected in the designated number of minutes. A message accompanies this shutdown. The workstation should immediately submit an `FPGetSrvrMsg` command to receive and display the message. This attention code can be used upon server shutdown (that is, when the administrator quits file service).
0100	The server is shutting down immediately, possibly due to an internal error, and can perform only minimal flushing. A message never accompanies this attention code.
1011	The server is shutting down, or the user will be disconnected in the designated number of minutes. A message accompanies this shutdown. The workstation should immediately submit an `FPGetSrvrMsg` command to receive and display the message. This is one of the codes used upon user disconnection (for example, when the administrator detects an intruder and disconnects him or her).
0100	The server is going down immediately (possibly because of an internal error) and can perform only minimal flushing. Number of minutes is ignored. No message ever accompanies such an attention code.

continued

Table 1-9 Valid combinations for the Attention Code bits (continued)

Combination	Meaning
0010	The server has a server message available for this workstation. The workstation should immediately submit an FPGetSrvrMsg command to receive and display the message. The extended bitmap is reserved for Apple Computer's use only.
0011	Reserved (AFP 2.1). Server Notification (AFP 2.2). The server is notifying the client of an event relating to the current session. Bit 0 in the extended bitmap indicates that the modification date of one of the volumes mounted from the server has changed. The client should issue an FPGetVolParms command for each volume mounted from the server.
0001	Reserved. The extended bitmap is reserved for Apple Computer's use only.
0000	Reserved. The extended bitmap is reserved for Apple Computer's use only.

Note that for some of the valid bit patterns, the lower 12 bits of AFPUserBytes are interpreted as the number of minutes before the action described by the bit pattern will take place. This value can be a number in the range 0 to 4094 ($FFE) inclusive. A value of 4095 ($FFF) means that the action is being canceled.

Some AFP 2.1–Related Questions and Answers

It appears to be a requirement that all user IDs be numerically different from all group IDs. When upgrading an old volume, must one change these IDs if they are not numerically different?

Yes. AppleShare's user ID numbers and group ID numbers have always been that way. In addition, AFP 2.1 servers must assign the guest user ID number 0 and the administrator/owner ID number 1.

Do FPMapID and FPMapName work the same way in AFP 2.1 as they do in AFP 2.0? (That is, must one choose the proper subfunction or get an error?)

Under AFP 2.1, calls to FPMapID must use subfunction code 1 or 2 and calls to FPMapName must use subfunction code 3 or 4. The subfunction used tells the call which database (user or group) to search first. This process doesn't affect the FPMapID command (since user and group IDs come from the same pool of numbers) except in one way: The user/group name will be returned for that ID no matter what. However, it does affect the FPMapName command. For example, if you have both a user and a group named "Fred" and you call FPMapName, the subfunction code will determine where the match is found (user or group).

Note that the AFP 2.1 server responds the same way for 1.1 and 2.0 clients as it does for AFP 2.1 clients.

On the Macintosh, PBGetCatInfo returns the file ID in the *ioDirID* field for files. Is this the value returned in the *FileNumber* field by FPGetFileDirParm?

The value returned in the *FileNumber* field by the AppleShare file server is what the file server gets from the Macintosh File Manager's PBGetCatInfo call. Since AppleShare implementations supporting AFP 2.1 on the Macintosh run under System 7, everything works as it would on a local volume. (That is, the value could represent a file ID or a directory ID, and you must use FPResolveID to check whether the value is a real file ID.)

How does the AFP file server know which directory is the Network Trash Folder?

The Network Trash Folder is identified by name and will not be localized in international versions of the Macintosh system software, as it is invisible.

Do servers using AFP 2.1 have to limit their icons to any particular size?

Yes, because Macintosh workstations running versions of AFP earlier than 2.1 behave poorly if the icon size is greater than 1536.

Is it true that the value of DTRefNum is the same as that of Volume ID for AFP desktop database calls?

Yes, but only if that volume has not been closed and then reopened. If it is reopened, new values for DTRefNum and Volume ID are assigned.

Is it true that FPCloseVol does not close all files open on a volume?

Yes, you should specifically close all open files on a volume before closing it, rather than relying on FPCloseVol to close them for you.

AFP over TCP

This section describes how the Transmission Control Protocol (TCP) can be used to transport AFP packets efficiently. With TCP as the transport protocol, AFP services can be made available over the Internet just as they are made over AppleTalk networks. When a user mounts a remote volume over TCP, the type of network over which the volume is mounted is completely transparent to the user. On local area networks, providing AFP services over TCP/IP effectively utilizes the bandwidth of high speed network media such as Fiber Distributed Data Interface (FDDI) and Asynchronous Transfer Mode (ATM).

TCP can be used as the transport protocol for AFP version 2.1 and version 2.2. In theory, versions of AFP prior to 2.1 could also use TCP as the transport protocol, but doing so is not recommended because the AFP 2.1 or later version of `FPGetSrvrInfo` is required to obtain a machine's IP address.

Implementation

A layer known as the Data Stream Interface (DSI) is used to provide AFP services over TCP. With minimal overhead, the DSI establishes an interface between AFP and TCP that is generic enough to be used over any data stream protocol. The DSI has the following characteristics:

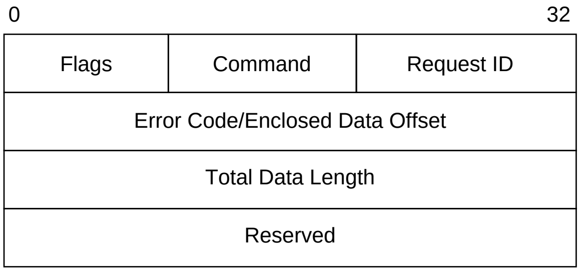
- It registers the AFP server on a well-known data stream port. For TCP, the port number is 548. Protocol suites that include a service-locating protocol can be used to advertise and locate an AFP server. For example, NBP can be used for AFP over ADSP, and the Service Advertisement Protocol (SAP) can be used for AFP over IPX/SPX.
- It uses a request/response model that supports multiple outstanding requests on any given connection. In other words, the request's window size may be greater than 1 in length.
- It replies to multiple outstanding requests in any order.
- It provides a one-to-one mapping between the AFP session and the port ID or connection ID maintained by the data stream protocol.
- It maintains some state information for every open client connection. This allows the server to demultiplex requests to an appropriate AFP session.

- It allows the AFP server to send and receive large packets. The size of the packets is based on the underlying network's maximum transmission unit (MTU).

The DSI Header

The DSI prepends the header shown in Figure 1-8 to every AFP request and reply packet.

Figure 1-8 DSI header format



Flags	Command	Request ID
Error Code/Enclosed Data Offset		
Total Data Length		
Reserved		

Table 1-10 describes each field in the DSI header.

Table 1-10 Fields in the DSI header

Field	Purpose
Flags	An 8-bit value that allows an AFP server to determine the packet type. The following packet types are defined: 0x00 = request 0x01 = reply
Command	An 8-bit value containing a value that represents a DSI command.

Table 1-10 Fields in the DSI header (continued)

Field	Purpose

Request ID	A 16-bit value containing a request ID on a per connection (session) basis. A request ID is generated by the host that issued the request. In reply packets, the request ID is used to locate the corresponding request.
	Request IDs must be generated in sequential order and can be from 0 to 65535 in value. The request ID after 65535 wraps to 0. The client generates the initial request ID and sends it to the server in a `DSIOpenSession` command. The server uses the following algorithm to anticipate the client’s next request ID:
	`if (LastReqID == 65536) LastReqID = 0;`
	`else LastReqID = LastReqID + 1;`
	`ExpectedReqID = LastReqID;`
	Servers begin generating request IDs at 0.

Error Code/ Enclosed Data Offset	In request packets, this field is ignored by the server for all commands except `DSIWrite`. For future compatibility, clients should set this field to zero for all commands except `DSIWrite`. In request packets for which the command is `DSIWrite`, this field contains a data offset that is the number of bytes in the data representing AFP command information. The server uses this information to collect the AFP command part of the packet before it accepts the data that corresponds to the packet. For example, when a client sends an `FPWrite` command to write data on the server, the enclosed data offset would be 12. In reply packets, this field contains an error code.
Total Data Length	A 32-bit unsigned value that specifies the total length of the data that follows the DSI header.
Reserved	A 32-bit field reserved for future use. Clients should set this field to zero.

continued

DSI Commands

DSI commands are similar to ASP commands, and they preserve all of the ASP commands except `ASPWriteContinue`. The DSI commands are listed in Table 1-11.

Table 1-11 DSI commands

Command name	Command code	Originator of command requests
`DSICloseSession`	1	Client and server
`DSICommand`	2	Client only
`DSIGetStatus`	3	Client only
`DSIOpenSession`	4	Client only
`DSITickle`	5	Client and server
`DSIWrite`	6	Client only
`DSIAttention`	8	Server only

Note
For consistency between ASP and DSI commands, the command code for `DSIAttention` is 8. ◆

DSIOpenSession

Usually, the `DSIOpenSession` command request is the first request issued by the client after it establishes a connection with an AFP server. (The client can also send a `DSIGetStatus` command request. In this case, the AFP server immediately tears down the connection after delivering the requested status information.) The `DSIOpenSession` command request opens a DSI session and delivers the client's initial request ID.

The data portion of a `DSIOpenSession` packet may contain options defined by the client (request) or server (reply). The options must conform to the format shown in Table 1-12.

Table 1-12 DSIOpenSession option format

0	8	16
Option Type	Option Length	Option

Table 1-13 describes each field in the option portion of the `DSIOpenSession` packet.

Table 1-13 Fields in the option portion of the DSIOpenSession packet

Field	**Purpose**
Option Type	An unsigned 8-bit value indicating the type of information contained by the *Option* field. Two types are defined: 0x00 = server request quantum. Sent by the server to the client to indicate that the *Option* field contains the size of the largest request packet the server can accept. 0x01 = attention quantum. Sent by the client to the server to indicate that the *Option* field contains the size of the largest attention packet the client can accept.
Option Length	An unsigned 8-bit value containing the length of the variable-length *Option* field that follows.
Option	A variable-length value sent in network byte order (most significant byte first) representing the number of bytes the server and the client can accept in request and attention packets, respectively, but not including the length of the DSI header and the AFP command. The length of the *Option* field is variable, but for maximum performance, it should be a multiple of 4 bytes.

DSICommand

Once the client opens a DSI session, the DSI is ready to accept and process `DSICommand` requests from the client. When it receives a `DSICommand` request, the DSI removes the header, saves the request context in its internal state, and passes the data (an AFP request) to the AFP server.

When the DSI receives a reply, it uses the *Command* and *RequestID* fields in the DSI header of the reply to match the reply with its corresponding request and request context in order to send the reply to the client. Once the DSI sends the reply to the client, the DSI reclaims storage allocated for the request context.

DSIWrite

The `DSIWrite` command request contains an `FPWrite` or an `FPAddIcon` request and the associated data. The amount of data to be written may be up to the size of the server request quantum described earlier in the section "DSIOpenSession" (page 34).

The AFP server may or may not be ready to accept the data, so the DSI only forwards the AFP request portion to the AFP server, using the enclosed data offset in the DSI header to determine the length of the AFP header.

Once it processes the header and determines that the client has the privileges required to write the data, the AFP server retrieves the data to be written from the DSI. Once the AFP server declines the request or the DSI finds that all of the data has been written, the DSI disposes of the data and reclaims the storage associated with it.

DSIAttention

The AFP server uses standard data stream packets to send `DSIAttention` command request packets to the client. The attention code is stored as part of the data in the DSI packet. The size of the attention code and any other attention type cannot be larger than the size specified by the attention quantum when the client opened the session. The default attention quantum size is 2.

DSITickle

The `DISTickle` command provides a way for AFP servers and clients to detect time-outs caused by the abnormal termination of DSI sessions and data stream connections. By default, an AFP server sends a `DSITickle` command request packet every 30 seconds to the client if the AFP server has not sent any other data to the client in the previous 30 seconds. Likewise, the client sends a `DSITickle` command request packet every 30 seconds to the client if the client server has not sent any other data to the AFP server in the previous 30 seconds.

If an AFP server does not receive any data from a client for two minutes, the AFP server terminates the session with the client. Likewise, the client terminates the session with the AFP server if the client does not receive any data from the server for two minutes.

Instead of using a timer to determine when to send a `DSITickle` command, many client implementations send a `DSITickle` command whenever they receive a `DSITickle` command from the AFP server.

DSICloseSession

To close a session, an AFP client or server sends a `DSICloseSession` command request. Without waiting for a reply, the sender of the `DSICloseSession` command closes the AFP session and reclaims all of the resources allocated to the session. Then it tears down the data stream connection.

Note
The `AFPLogout` command does not close the session. ◆

DSIGetStatus

In the context of data stream communication, the client must establish a session with the server in order to exchange information with it, but in the context of ASP, a client can send an `ASPGetStatus` command to the server without establishing a session. To support `ASPGetStatus`, the AFP server supports the `DSIGetStatus` command on its listening port.

To obtain ASP status information, the client must establish a connection on the server's listening port. The client then sends a `DSIGetStatus` command to the server. The server then returns the status information to the client and immediately tears down the connection.

Apple Filing Protocol Reference

This chapter describes commands that changed in AFP 2.1, 2.2, and 3.0 or that were added to AFP for those versions. For a discussion of AFP 2.0, see *Inside AppleTalk*, Second Edition.

Table 2-3 lists the commands that were added for AFP version 3.0.

Table 2-1 Commands added for AFP version 3.0

Command	Constant	Hexadecimal	Decimal
FPByteRangeLockExt	kFPByteRangeLockExt	0x003B	59
FPReadExt	kFPReadExt	0x003C	60
FPWriteExt	kFPWriteExt	0x003D	61
FPGetAuthMethods	kFPGetAuthMethods	0x003E	62
FPLoginExt	FPLoginExt	0x003F	63

Table 2-2 lists the commands that were modified for AFP version 2.2 and later.

Table 2-2 Commands modified for AFP version 2.2 and later

Command	Modification
FPGetSrvrInfo	Returns information about a server’s support for TCP/IP.
FPGetVolParms	Returns information about volumes greater than 4 gigabytes (GB) in size.

Table 2-3 lists the commands that were added for AFP version 2.1 and later. Each command code is a 16-bit integer sent high-byte first in the packet.

Table 2-3 Commands added for AFP version 2.1 and later

Command	Constant	Hexadecimal	Decimal
`FPGetSrvrMsg`	`kFPGetSrvrMsg`	0x0026	38
`FPCreateID`	`kPCreateID`	0x0027	39
`FPDeleteID`	`kFPDeleteID`	0x0028	40
`FPResolveID`	`kFPResolveID`	0x0029	41
`FPExchangeFiles`	`kFPExchangeFiles`	0x002A	42
`FPCatSearch`	`kFPCatSearch`	0x002B	43

AFP Commands

This section describes AFP commands that have been added or modified since AFP 2.0. For a discussion of AFP 2.0, see *Inside AppleTalk*, Second Edition.

FPByteRangeLockExt

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

Inputs	*flags* (int)	Flags indicating whether the offset field is relative to the beginning or end of the fork when locking a range and indicating whether the range is being locked or unlocked.
	forkRef (int)	Open fork reference number.
	offset (16 bytes)	Offset to the first byte of the range to be loocked or unlocked (can be negative if `flags` is set to `end`).
	length (16 bytes)	Number of flags to be locked or unlocked (a signed, positive 16-byte integer; cannot be negative except for the special value $FFFFFFFFFFFFFFFF).
Outputs	*FPError* (long)	
	RangeStart (16 bytes)	Number of the first byte of the range that was just locked; this number is valid only when returned from a successful command to lock a range
Result codes	`afpParmErr`	Open fork refnum is unknown; a combination of the flags and offset fields specifies a range that starts before byte zero.
	`afpLockErr`	Some or all of the requested range is locked by another user.
	`afpNoMoreLocks`	Server's maximum lock count has been reached.
	`afpRangeOverlap`	User tried to lock some or all of a range that the user has already locked.
	`afpRangeNotLocked`	User tried to unlock a range that was locked by another user or not locked at all.

VERSION

Supported in AFP 3.0 and later.

DISCUSSION

The FPByteRangeLockExt command locks or 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.

Bytes are numbered starting from 0. The latter value is the maximum size of the fork. The end of fork (end of file in Macintosh terminology) is one greater 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 of zero, flags to Start, and length to $FFFFFFFFFFFFFFFF locks the entire for to the maximum size of the fork. Specifying an offset other than zero, flags to Start, and length to $FFFFFFFFFFFFFFFF locks a range beginning at offset and extending to the maximum size of the fork.

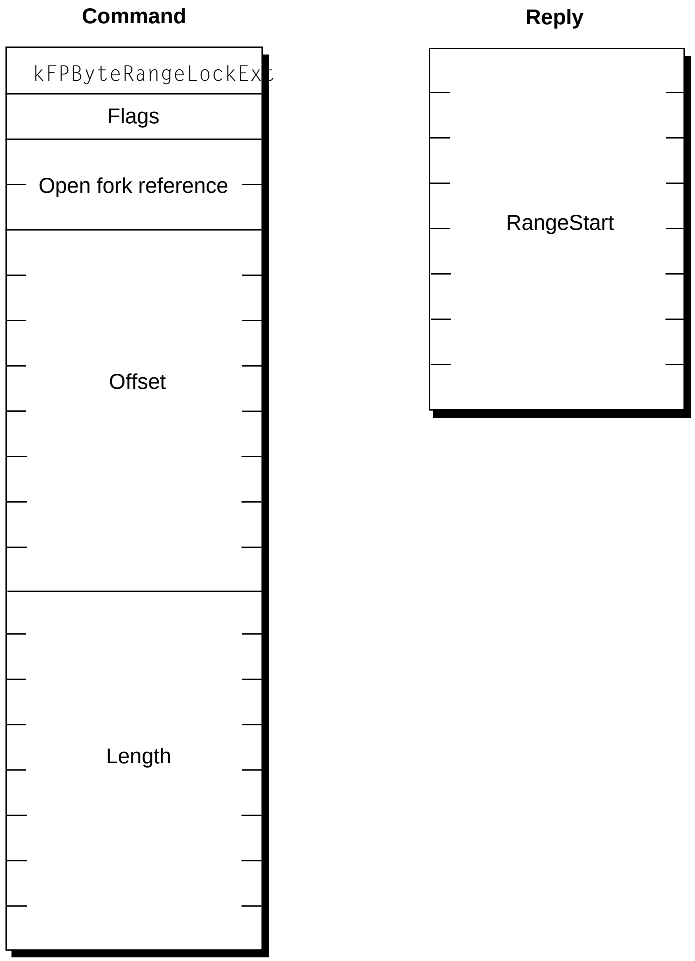
Setting the flags field to End allows a lock to be offset relative to the end of the fork. This enables a user to set a lock when the user does not know the exact end of the fork, as can happen when multiple writers are currently modifying the fork. The server returns the number of the first locked byte.

Lock conflicts are determined by the value of forkRef. 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 RangeNotLocked result code if a user tries to unlock a range that was locked by another user or that was not locked at all.

You cannot unlock part of range. To unlock a range, flags must be set to Start, the length field must match the size of the range that was locked, and the offset parameter must match the number of the first byte in the locked range. If the range was locked with the flags set to Start, use the same value of offset to unlock the range that was used to lock the range. If the range was locked with flags set to End, set offset to the value of RangeStart that was returned by the server.

Figure 2-1 Command and reply blocks for the FPRangeLockExt command



FPCatSearch

Searches a volume for files that match specified criteria.

Inputs	*VolumeID* (int)	The ID of the volume on which the file is located.
	ReqMatches (long)	The maximum number of matches to return.
	Reserved (long)	Reserved. (Must be zero.)
	CatPosition (16 bytes)	Current position in the catalog.
	FileRsltBitmap (int)	The fields in the `File` parameter that are to be returned; this field is the same as the File Bitmap field in the `FPGetFlDrParms` command (with some restrictions, explained later in this section).
	DirRsltBitmap (int)	The fields in the `Dir` parameter that are to be returned; this field is the same as the Directory Bitmap field in the `FPGetFlDrParms` command (with some restrictions, explained later in this section).
	RequestBitmap (long)	The fields in the `File` and `Dir` parameters that are to be searched. (The structure of the bitmap is shown later in this section.)
	Specification1	Search criteria lower bounds and values.
	Specification2	Search criteria upper bounds and masks.
Outputs	*CatPosition* (16 bytes)	Current position in the catalog.
	FileRsltBitmap (int)	Copy of the input bitmap.
	DirRsltBitmap (int)	Copy of the input bitmap.
	ActualCount (long)	The number of matches that were actually found.
	Results	An array of records describing the matches that were found.
	FPError (long)	

Result codes	`afpCallNotSupported`	The AFP version is earlier than 2.1.
	`afpCatalogChanged`	The catalog has changed and *CatPosition* may be invalid. No matches were returned.
	`afpParmErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.
	`afpEofError`	No more matches.

VERSION

Supported by AFP 2.1 and later.

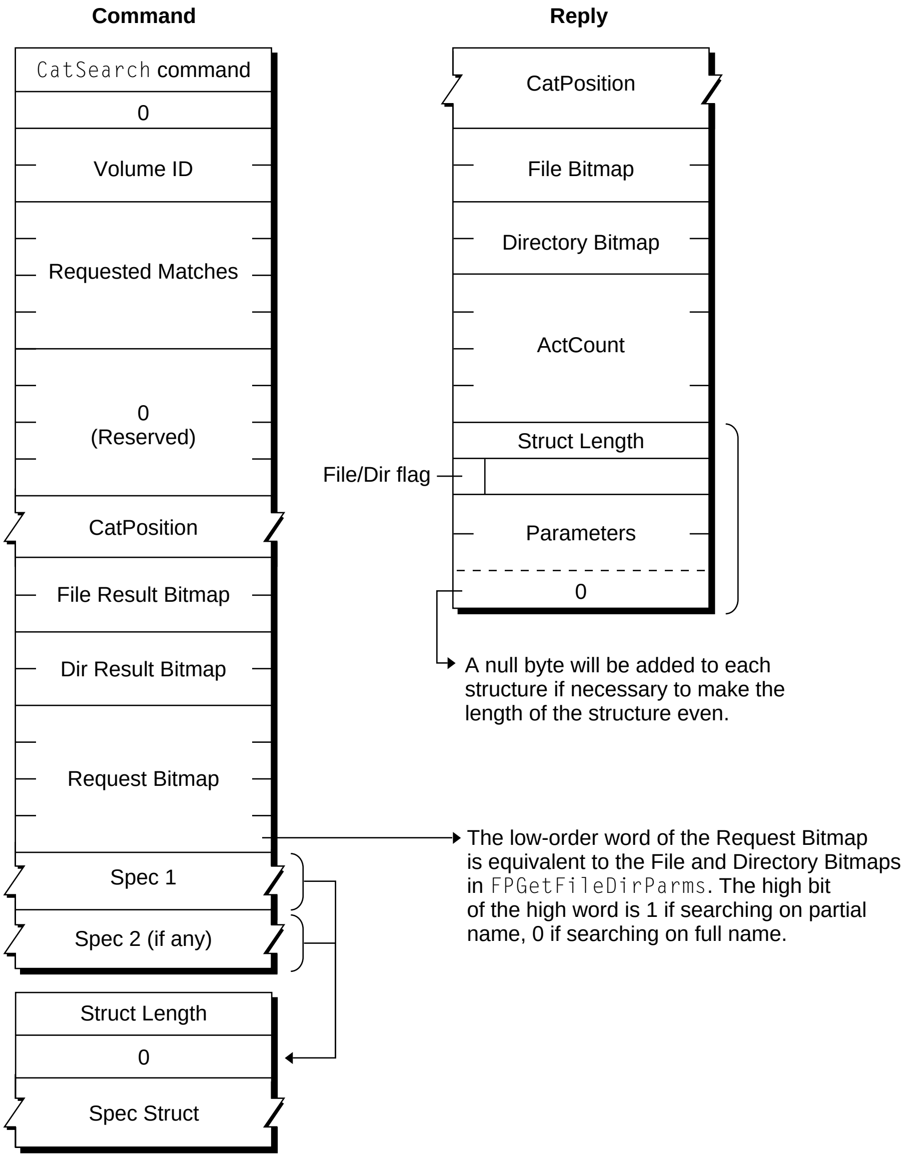
DISCUSSON

The `FPCatSearch` command searches a volume for files that match a specified criteria and returns an array of records that describes the matches that were found.The criteria can include any fields in the file bitmaps, directory bitmaps, or both, that are defined for the `FPGetFileDirParms` command. Information parameters for the matching files and directories are returned. These parameters can also be any of those specified for the `FPGetFileDirParms` command.

Before issuing this command, the user must call `FPOpenVol` for the volume that is to be searched.

Figure 2-2 shows the command and reply blocks for the `FPCatSearch` command.

Figure 2-2 Command and reply blocks for the FPCatSearch command


Command
CatSearch command
0
Volume ID
Requested Matches
0
(Reserved)
CatPosition
File Result Bitmap
Dir Result Bitmap
Request Bitmap
Spec 1
Spec 2 (if any)
Struct Length
0
Spec Struct
Reply
CatPosition
File Bitmap
Directory Bitmap
ActCount
Struct Length
File/Dir flag
Parameters
0
A null byte will be added to each structure if necessary to make the length of the structure even.
The low-order word of the Request Bitmap is equivalent to the File and Directory Bitmaps in FPGetFileDirParms. The high bit of the high word is 1 if searching on partial name, 0 if searching on full name.

The first word of the *CatPosition* parameter specifies whether the field denotes a "real" catalog position or a hint. If the first word is zero, FPCatSearch starts the search at the beginning of the volume. If the first word is nonzero, *CatPosition* is a "real" catalog position and FPCatSearch begins its search 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 request bitmap. All variable-length parameters (such as those containing names) are put at the end of each specification record. An offset is stored in the 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:

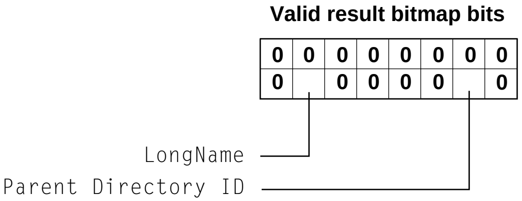
- In the name field, *Specification1* holds the target string; *Specification2* must always have a nil name field.
- In all date and length fields, *Specification1* holds the lowest value in the target range and *Specification2* holds the highest value in the target range.
- In file attributes and Finder 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.

The FPCatSearch command returns the error afpEofError 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 make a request for six (10 minus 4) more matches, using the same *CatPosition* value that was received in the previous reply. This process continues until the originally requested matches are received or an afpEofError is returned. If FPCatSearch returns the error afpCatalogChanged, the client cannot continue the search. The client must restart the search by setting the first word of *CatPosition* to zero.

The FPCatSearch command returns files or directories or both, depending on the *FileRsltBitmap* and *DirRsltBitmap* fields. If the *FileRsltBitmap* field is zero, FPCatSearch assumes that you are not searching for files. Likewise, if the DirRsltBitmap field is zero, FPCatSearch assumes that you are not searching for directories. If both fields are nonzero, FPCatSearch returns both files and directories. Note that if you are searching for both files and directories, certain restrictions apply as to what fields FPCatSearch will search. The rest of this section describes these restrictions.

The only valid bits for the *FileRsltBitmap* and *DirRsltBitmap* fields are the `LongName` and `Parent Directory ID` bits. Figure 2-3 shows the valid Result Bitmap bits.

Figure 2-3 Valid result bitmap bits



The low-order word of *RequestBitmap* is roughly equivalent to the File and Directory Bitmaps in `FPGetFileDirParms`. (See the bitmaps for the differences.) The high bit of the high-order word of *RequestBitmap* 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 2-4 shows the valid directory bits. The `FPCatSearch` command can only search for this information when it searches for directories.

Figure 2-4 Valid directory bits

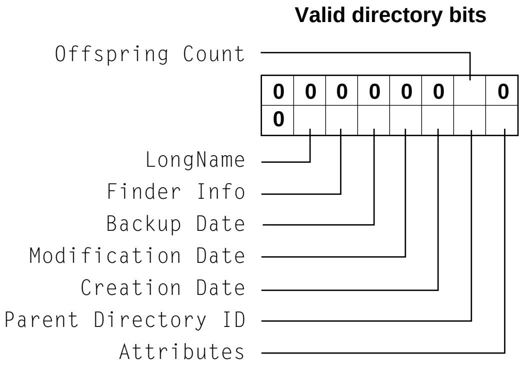


Figure 2-5 shows the valid file bits. The FPCatSearch command can search for this information only when it searches for files.

Figure 2-5 Valid file bits

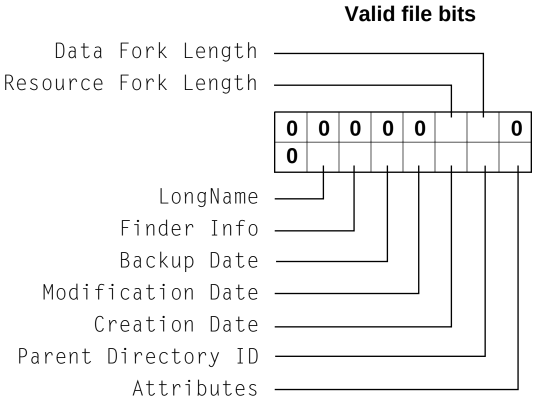
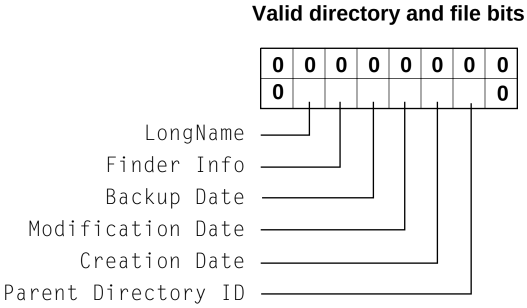


Figure 2-6 shows the valid directory and file bits. The FPCatSearch command can search for this information when it searches for directories and files.

Figure 2-6 Valid directory and file bits



The inhibit bits are the only valid bits that the FPCatSearch command can search for in the *Attributes* parameter. For files, these bits are DeleteInhibit, RenameInhibit, and WriteInhibit. For directories, these bits are DeleteInhibit and RenameInhibit. You cannot search any bits in *Attributes* when you are searching for files and directories.

PRIVILEGES

The user need have no special access privileges to use this command; however, to see all the files, folders, or files and folders that match the specified criteria, the user must have Read Only or Read & Write privileges to them. The FPCatSearch command skips folders for which the user does not have Read Only or Read & Write privileges.

FPCreateID

Creates a unique file ID for a specified file.

Inputs	*VolumeID* (int)	The ID of the volume on which the file ID is to be created.
	DirectoryID (long)	The ID of the directory in which the file is to be created.
	PathType (byte)	Path type of the pathname: 0 = short name 1 = long name
	Pathname (string)	String name of the file that is the target of the file ID (that is, the filename of the file for which you want to create the file ID).
Outputs	*FileID* (long)	File ID that was created for the specified file.
	FPError (long)	
Result codes	`afpCallNotSupported`	The AFP version is earlier than 2.1.
	`afpObjectNotFound`	The target file does not exist.
	`afpIDExists`	A file ID already exists for this file. The file ID is returned in the FileID field.
	`afpObjectTypeErr`	Object defined was a directory, not a file.
	`afpVolLocked`	The destination volume is read-only.
	`afpAccessDenied`	User does not have the privileges required to issue this command.
	`afpParmErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

VERSION

Supported by AFP 2.1 and later.

DISCUSSON

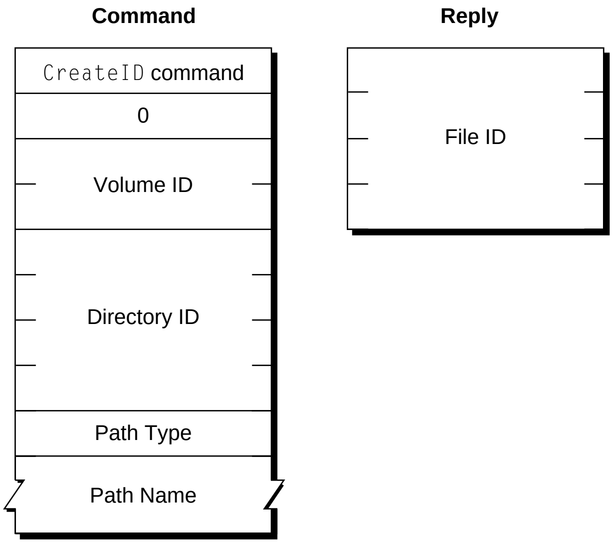
File IDs provide a means of keeping track of a file even if its name or location changes. The scope of file IDs is limited to the files on a volume. File IDs cannot be used across volumes.

Before using this command, the client must have called `FPOpenVol` for this volume.

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.

Figure 2-7 shows the command and reply blocks for the `FPCreateID` command.

Figure 2-7 Command and reply blocks for the FPCreateID command



PRIVILEGES

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

FPDeleteID

Invalidates all instances of the specified file ID.

Inputs	*VolumeID* (int)	The ID of the volume on which the file ID is to be deleted.
	FileID (long)	The file ID that is to be invalidated.
Outputs	*FPError* (long)	
Result codes	`afpCallNotSupported`	The AFP version is earlier than 2.1.
	`afpObjectNotFound`	The target file does not exist. The file ID is deleted anyway.
	`afpIDNotFound`	File ID was not found. (No file thread exists.)
	`afpObjectTypeErr`	Object defined was a directory, not a file.
	`afpVolLocked`	The destination volume is read-only.
	`afpAccessDenied`	User does not have the privileges required to use this command.
	`afpParmErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

VERSION

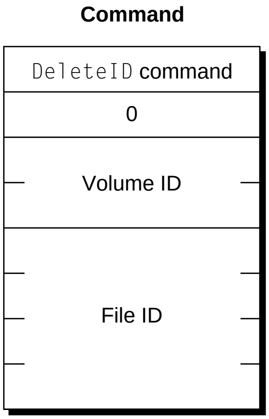
Supported by AFP 2.1 and later.

DISCUSSION

Before using this command, the user must have called `FPOpenVol` for this volume.

Figure 2-8 shows the command block for the `FPDeleteID` command.

Figure 2-8 Command block for the FPDeleteID command



PRIVILEGES

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

FPExchangeFiles

Preserves existing file IDs when an application performs a Save or a Save As operation.

Inputs	*VolumeID* (int)	The ID of the volume on which the two files are located.
	SrcDirID (long)	The ID of the directory that contains the source file.
	DestDirID (long)	The ID of the directory that contains the destination file.
	SrcPathType (byte)	Path type of the source pathname: 1 = short name 2 = long name
	SrcPathName (string)	String name of the source file.
	DestPathType (byte)	Path type of the source pathname: 1 = short name 2 = long name
	DestPathName (string)	String name of the destination file.
Outputs	*FPError* (long)	
Result codes	`afpCallNotSupported`	The AFP version is earlier than 2.1.
	`afpIDNotFound`	File ID was not found. (No file thread exists.)
	`afpObjectTypeErr`	Object defined was a directory, not a file.
	`afpBadIDErr`	File ID number is not a defined file ID.
	`afpAccessDenied`	User does not have the privileges required to use this command.
	`afpParmErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

VERSION

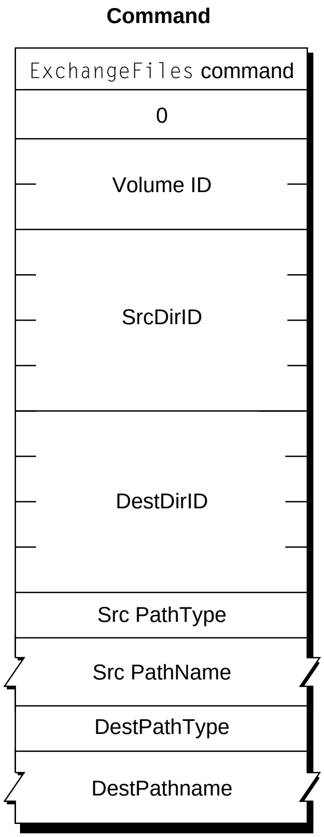
Supported by AFP 2.1 and later.

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. Before you call `FPExchangeFiles`, you must call `FPOpenVol` for the volume on which the files reside.

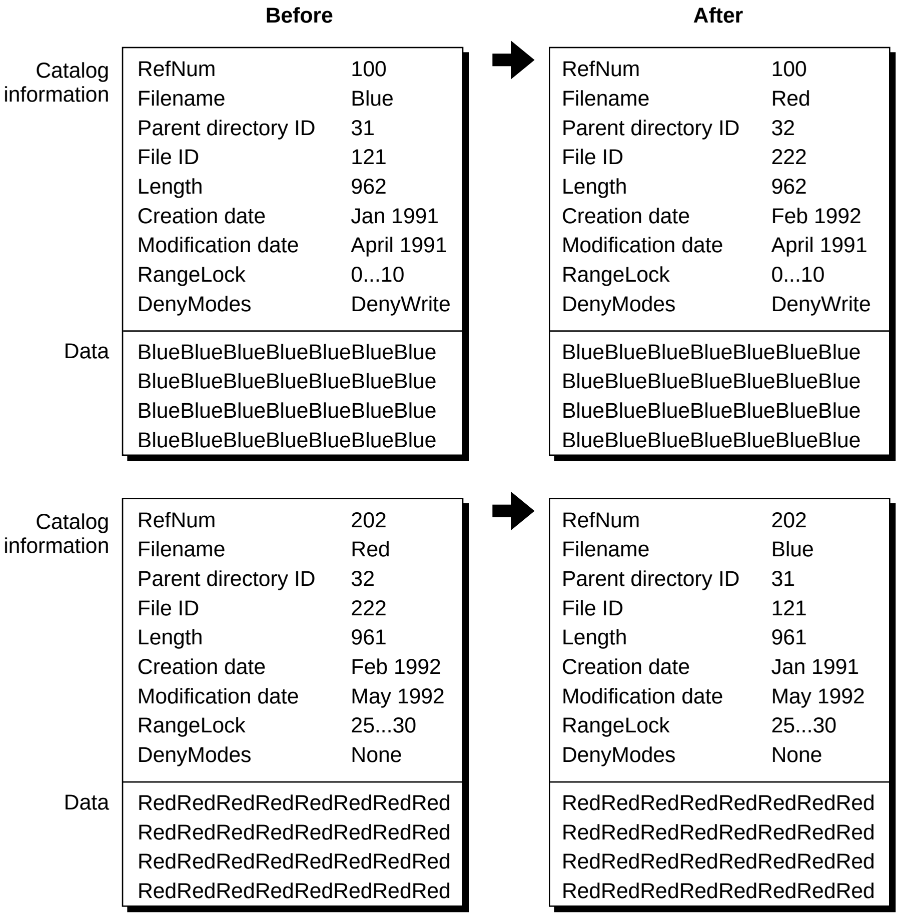
Figure 2-9 shows the command block for the `FPExchangeFiles` command.

Figure 2-9 Command block for the FPExchangeFiles command



The following example shows the results of an `FPExchangeFiles` operation between two files named Blue and Red.

Figure 2-10 Example of calling FPExchangeFiles



	Before		After	
Catalog information	RefNum	100	RefNum	100
	Filename	Blue	Filename	Red
	Parent directory ID	31	Parent directory ID	32
	File ID	121	File ID	222
	Length	962	Length	962
	Creation date	Jan 1991	Creation date	Feb 1992
	Modification date	April 1991	Modification date	April 1991
	RangeLock	0...10	RangeLock	0...10
	DenyModes	DenyWrite	DenyModes	DenyWrite
Data	BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue		BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue BlueBlueBlueBlueBlueBlueBlue	

	Before		After	
Catalog information	RefNum	202	RefNum	202
	Filename	Red	Filename	Blue
	Parent directory ID	32	Parent directory ID	31
	File ID	222	File ID	121
	Length	961	Length	961
	Creation date	Feb 1992	Creation date	Jan 1991
	Modification date	May 1992	Modification date	May 1992
	RangeLock	25...30	RangeLock	25...30
	DenyModes	None	DenyModes	None
Data	RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed		RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed RedRedRedRedRedRedRedRed	

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.

PRIVILEGES

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

FPGetAuthMethods

Gets the supported authentication methods for a directory.

VERSION

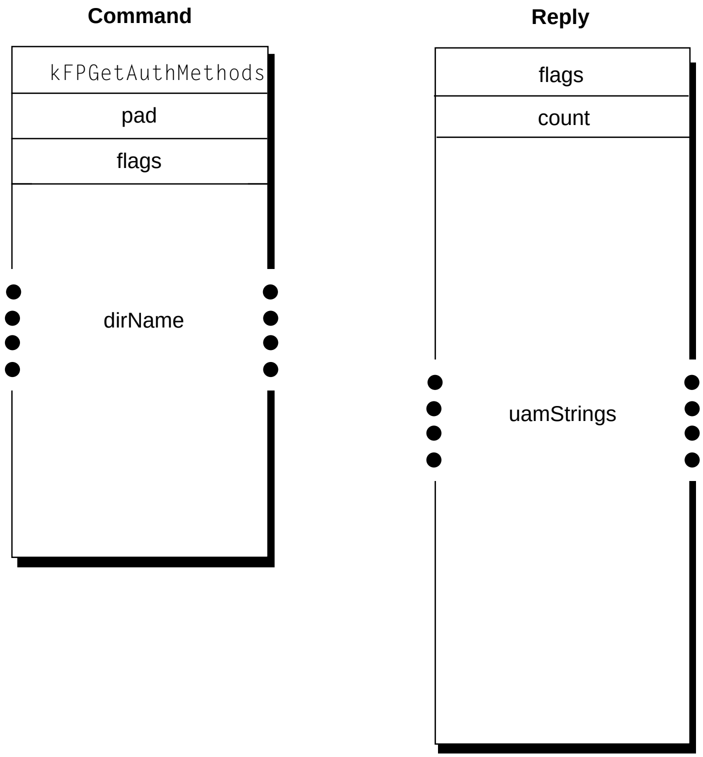
Supported in AFP 3.0 and later.

DISCUSSION

The `FPGetAuthMethods` command gets the authentication methods for the specified directory.

Inputs	*pad* (uchar)	Pad byte.
	flags (unsigned short)	Flags providing additional information. (none are defined yet)
	dirname (AFPName)	An `AFPName` structure specifying the path type and the path for which for which authentication methods are to be obtained. The path type is a one-byte value that can be one of `kFPShortName`, `kFPLongName`, or `kFPUTF8Name`. When the path type is `kFPShortName` or `KFPLongName`, the Pascal string is a one-byte value specifying the length of the pathname that follows. When the path type is `kFPUTF8Name`, the Pascal string is a two-byte value specifying the length of the pathname that follows.
Outputs	*FPError* (long)	
	flags (unsigned short)	Flags providing additional information. (none are defined yet)
	count (uchar)	A one-byte value specifying the number of authentication methods that follow.
	uamStrings (packed Pascal strings)	Packed Pascal strings containing the names of the authentication methods.
Result codes	`afpObjectNotFound`	The specified authentication method could not be found.

Figure 2-11 Command and reply blocks for the FPGetAuthMethods command



FPGetSrvrInfo

Retrieves information about a server.

Inputs	*SAddr* (EntityAddr)	Internet address of the server. (OT Address).
Outputs	*FPError* (long)	
	Flags (long)	Flags describing capabilities of the server, consisting of: 0 = supports copy file. 1 = supports changing passwords. 2 = doesn’t allow passwords to be saved. 3 = supports server messages. 4 = supports server signature. 5 = supports TCP/IP. 6 = supports server notifications.
	ServerName (string)	A string containing the name of the server.
	MachineType (string)	A string containing a description of the server’s hardware, operating system, or both.
	AFPVersions (string)	A string containing the versions of AFP that the server supports.
	UAMs(string)	A string containing the UAMs that the server supports.
	VolumeIconAndMask (256 bytes)	128 bytes of icon data and 128 bytes of mask data.
	ServerSignature (16 bytes)	A 16-byte number that uniquely identifies the server.
	NetworkAddresses (AFP Network Address)	The server’s network addresses. (The AFP Network Address format is described later in this section.)
Result codes	`afpNoServer`	The server is not responding.

VERSION

Modified for AFP 2.2 and later.

DISCUSSION

The FPGetSrvrInfo command retrieves information about the server in the form of an information block.

Note

FPGetSrvrInfo is the only AFP command that the client can use without establishing a session with the server. ◆

To facilitate access to all of the fields in the information block, the block begins with a header containing the offset to each field in the block: first an offset to the machine type, followed by the offset to the AFP versions strings, the offset to the UAM strings, the offset to the volume icon and mask, the flags word, the server name padded to an even boundary, the offset to the server signature, and the offset to the IP numbers. The volume icon and mask, server signature, and IP numbers are optional. If the volume icon and mask is not included, the offset is zero. The offsets for the server signature and IP numbers are included only if either of their bits in the flags word are set.

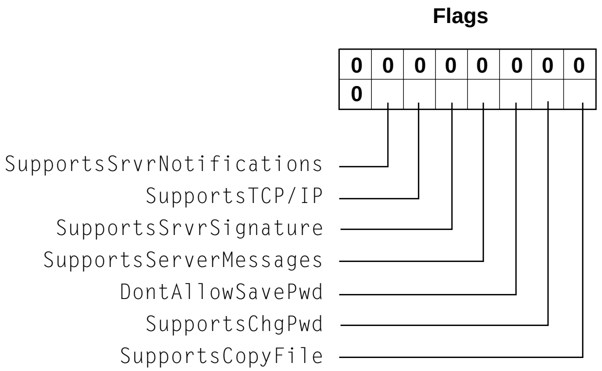
Because the server can pack the fields in the information block in any order, no assumptions should be made about the order of the fields; instead applications should access the fields only through the offsets. The exception is the *ServerName* field, which is always after the *Flags* field.

The AFP version and UAM strings are each formatted as one byte containing the number of strings that follow, with the strings packed back-to-back without padding. AFP version 2.2 is denoted by the string “AFP2.2”, and AFP version 2.1 is denoted by the string “AFPVersion 2.1”.

If the *VolumeIconAndMask* field is not included, its offset is zero.

Figure 2-12 shows the bits in the *Flags* field.

Figure 2-12 Flags field in the FPGetSrvrInfo information block



Offsets for the *ServerSignature* and *NetworkAddresses* fields are present only if either of their bits in the *Flags* field is set.

The *ServerSignature* field contains a unique identifier for the server. Client applications should use the server signature to ensure that the client does not log on to the same server multiple times. Preventing multiple logons is important when the server is configured for multihoming.

The *NetworkAddresses* field contains a list of addresses that the client can use to connect to the server over AppleTalk or TCP/IP. Each address is stored as an AFP Network Address. The format of an AFP Network Address is shown in Figure 2-13.

Figure 2-13 AFP Network Address format

Length	Tag	Address

Each AFP Network Address data item consists of a length byte containing the total length in bytes of the data item, followed by a tag byte identifying the type of address contained by the *Address* field, followed by up to 254 bytes of data. Table 2-4 lists the possible values of the *Length* and *Tag* fields and describes the type of address stored in the *Address* field.

Table 2-4 Fields of the AFP network address format

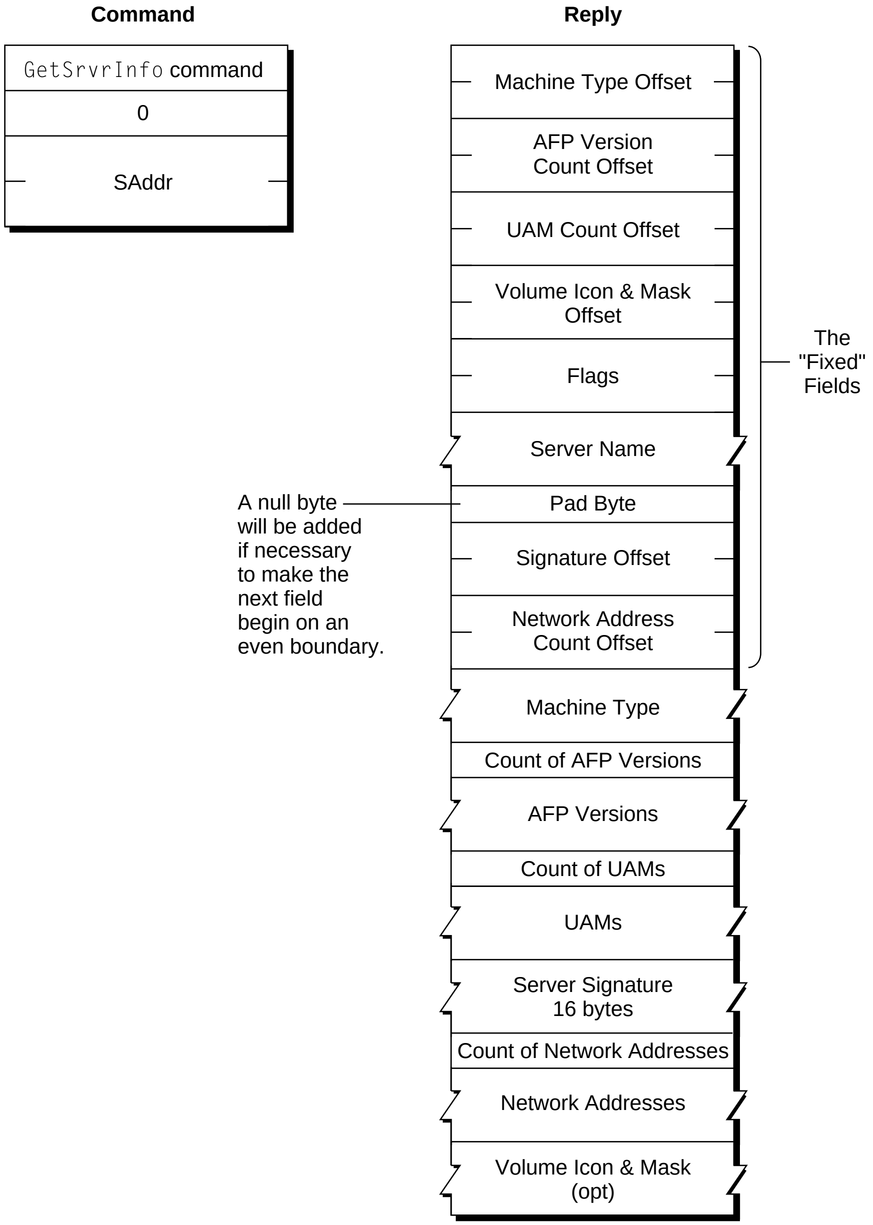
Total length in bytes	**Tag**	**Address**
06	0x01	IP address consisting of 4 bytes.
08	0x02	IP address (4 bytes) with port number (2 bytes).
06	0x03	DDP address (2 bytes for the network number, 1 byte for the node number, and 1 byte for the socket number.

The network address format provides the available network address to the client. Tags that the client does not recognize must be ignored.

Note
Tag 0x00 and 0x04 to 0x40 are reserved. ◆

Figure 2-14 shows the command and reply blocks for the `FPGetServrInfo` command.

Figure 2-14 Command and reply blocks for the FPGetSrvrInfo command



FPGetSrvrMsg

Gets a string message from the server.

Inputs	*MsgType* (int)	Type of server message: 0 = logon 1 = server (This value should be used in response to the Server Message bit in the attention code.)
	MsgBitmap (int)	Bitmap indicating what information to pass with the server message. (Currently, this is only the message string itself.) The structure of the bitmap is shown later in this section.
Outputs	*MsgType* (int)	Type of server message: 0 = logon 1 = server
	MsgBitmap (int)	Bitmap indicating what information was passed.
	SrvrMessage (string)	String message from the server.
	FPError (long)	
Result codes	`afpCallNotSupported`	The server does not implement `FPGetSrvrMsg`, or the AFP version is earlier than 2.1.
	`afpUserNotAuth`	The user was not logged on.
	`afpBitMapErr`	The specified bitmap has unrecognized bits set.

VERSION

Supported by AFP 2.1 and later.

DISCUSSION

The client uses the FPGetSrvrMsg command to receive shutdown, user, and logon messages from the server. Usually, the server sends an attention code to the client when these messages are available. However, the client can call FPGetSrvrMsg at any time. The server returns an empty or zero-length string if no message is available.

The logon message type allows the server to send a message to a client at logon time. The client can query the server for a logon message at logon time, or whenever it is convenient to do so. If there is no logon message, FPGetSrvrMsg returns a zero-length string, and nothing need be displayed.

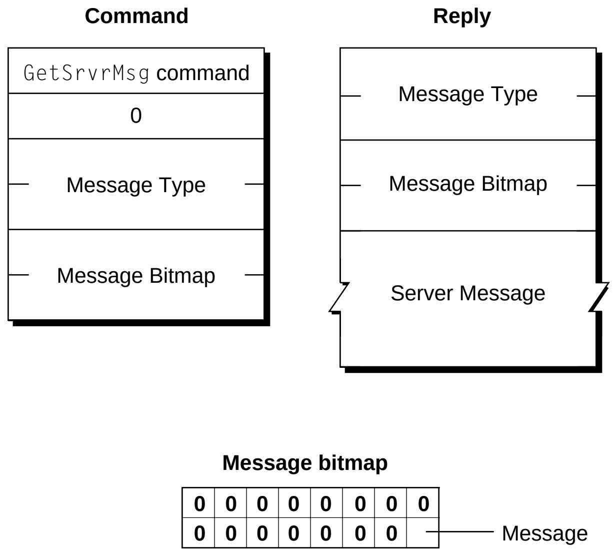
There are two server message types:

- Shutdown. In addition to sending an attention code when the server is going to shut down, the server can send a message explaining, for example, why the server is going down, how long it will be down, and so on. The client is made aware that a shutdown message is available by the server’s setting the Server Message bit in AFPUserBytes along with the Shutdown bit.
- User. The server can send a message to a specified user or users. The client is made aware that a user message is available when the server sets the Server Message bit in AFPUserBytes. Clients that implement older AFP versions should ignore this bit.

The maximum size of any of these messages is 200 bytes including the length byte (a Str199). The attention mechanism currently being used has been augmented to let the client know that there is a server message. The client then requests (by means of FPGetSrvrMsg) the message from the server.

Figure 2-15 shows the command and reply blocks for the FPGetSrvrMsg command.

Figure 2-15 Command and reply blocks for the FPGetSrvrMsg command



PRIVILEGES

The user must be logged on to the server to receive server message notifications. Other than that, the user need have no special access privileges to use this command.

FPGetVolParms

Retrieves parameters that describe a specified volume.

Inputs	*SRefNum* (short)	Session reference number.
	VolumeID (short)	Volume ID for the volume whose parameters are to be retrieved.
	Bitmap (short)	Bitmap describing the parameters that are to be returned. The bits are interpreted as follows: 0 = attributes (short) consisting of the following flag: 0 = ReadOnly 1 = signature 2 = creation date (long) 3 = modification date (long) 4 = backup date (long) 5 = volume ID (short) 6 = bytes free (unsigned long) 7 = bytes total (unsigned long) 8 = volume name (short) 9 = extended bytes free (8 bytes 10 = extended bytes total (8 bytes) 11 = allocation block size (4 bytes in network order)
Outputs	*FPError* (long)	
	Bitmap (short)	Copy of input parameter.
	Requested parameters	
Result codes	`afpParmErr`	Session reference number or volume identifier is unknown.
	`afpBitmapErr`	The specified bitmap has unrecognized bits set.

VERSION

Modified for AFP 2.2 and later.

DISCUSSION

The `FPGetVolParms` command retrieves parameters that describe a volume as specified by the volume's volume ID. Before you can call `FPGetVolParms`, you must call `FPOpenVol` for the volume.

Note
For AFP 2.2, `FPOpenVol` and `FPGetVolParms` use the VolParms bitmap. ◆

The server responds to the `FPGetVolParms` command by returning a reply block containing a bitmap for the volume parameters and the parameters themselves. All variable-length parameters, such as the *VolumeName* field, 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 fields. The variable-length fields are then packed after all fixed-length fields.

The Extended Bytes Free and Extended Bytes Total parameters are intended for use with volumes that are more than 4 GB in size. If a volume is more than 4 GB, the Bytes Free and Bytes Total parameters may not reflect the actual values. When that is the case, the Bytes Total parameter reflects the maximum value the volume can contain minus 4 GB, and the Bytes Free parameter reflects the bytes free up to a maximum of 4 GB. In any case, Extended Bytes Free and Extended Bytes Total always reflect the correct values.

Note
The Extended Bytes Free and Extended Bytes Total parameters are returned in network byte order (most significant byte first). ◆

FPLoginExt

Establishes an AFP session with a server.

Inputs	*pad* (uchar)	Pad byte.
	flags (unsigned short)	Flags providing additional information. (none are defined yet)
	authMethod (UAMString)	One or more Pascal strings (`Str16`) containing the names of authentication methods.
	userName (AFPName)	An `AFPName` structure specifying the user's name. Not required if authMethod is set to `no user auth`.
	dirName (AFPName)	An `AFPName` structure specifying the login directory for the user specified by `userName`. Not required if authMethod is set to `no user auth`.
	padToEven (uchar)	A pad byte. Not required if authMethod is set to `no user auth`.
	AuthInfo (uchar)	Information required by the authentication method, such as a password. Not required if authMethod is set to `no user auth`.
Outputs	*FPError* (long)	Server is not responding.
	SRefNum (int)	Session reference number used to refer to this session in all subsequent calls (valid if no error or `AuthContinue` are returned as the result code).
	ID (int)	An ID to be passed to the `FPLoginCont` call (valid only if `AuthContinue` is returned as the result code).
	userAuthInfo	A value returned by certain UAMs (valid only if `AuthContinue` is returned as the result code).
Result codes	*NoServer*	Server is not responding.
	BadVersNum	

BadUAM	The specified authentication method is unknown.
ParamErr	The specified user is unknown.
UserNotAuth	Authentication failed.
AuthContinue	Authentication is not yet complete.
ServerGoingDown	Server is shutting down.
MiscErr	User is already authenticated.

VERSION

Supported in AFP 3.0 and later.

DISCUSSION

The `FPLoginExt` command establishes an AFP session with a server. The client sends the server the authentication method to use (obtained by calling `FPGetAuthMethods` (page 59).

When the Cleartext Password UAM (`Cleartxt Passwrd`) is used, the user'sname and password are sent in the `authInfo` field. The password is transmitted in cleartext and must be padded with null bytes to make it 8 bytes in length. The server looks up the password for that user and compares it to the password in the command block. If the two passwords match, the user has been authenticated and the login succeeds. If they do not match, an `afpUserNotAuth` result code is returned.

When the Random Number Exchangage UAM (`Randnum Exchange`) is used, only the user name is sent in the `authInfo` field. If the user name is valid, the server generates an 8-byte random number and sends it back to the client, along with a ID number and an `AuthContinue` result code. The `AuthContinue` result code indicates that all is well at this point, but the user has not yet been authenticated.

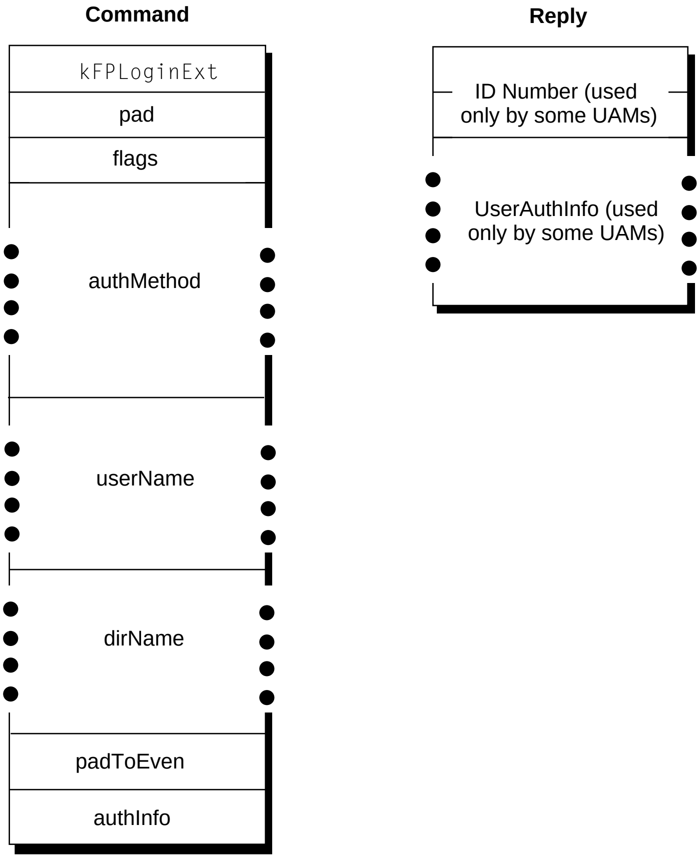
The client then uses the password as a key to encrypt the random number and sends the result to the server in the `userAuthInfo` field of the `FPLoginCont` command along with the ID number returned by the server. The server uses the ID number to associate the previous `FPLoginExt` command with this call to `FPLoginCont`. The server looks up the password for that user and uses it as a key to encrypt the same random number. If the two encrypted numbers match, the user has been authenticated and the login succeeds. Otherwise, the server returns a `UserNotAuth` result code.

If the server returns any result code other than AuthContinue, the session has not be established.

User name comparison is case-insensitive and diacritical-sensitive; password comparison is case-sensitive.

Random-number encryption is performed using DES.

Figure 2-16 Command and reply blocks for the FPLoginExt command



FPReadExt

Reads a block of data from an open fork.

Inputs	*pad* (int)	Pad byte.
	forkRef (int)	Open fork reference number.
	offset (16 bytes)	Number of the first byte to be read.
	reqCount (16 bytes)	Number of bytes to be read.
Outputs	*FPError* (long)	
	ActCount (16 bytes)	Number of bytes actually read from the fork.
	Requested data	
Result codes	`afpParmErr`	Session reference number or open fork reference number is unknown; `reqCount` or `offset` is negative.
	`afpAccessDenied`	Fork was not opened for read access.
	`afpEofError`	End of fork was reached.
	`afpLockErr`	Some or all of the requested range is locked by another user.

VERSION

Supported in AFP 3.0 and later.

DISCUSSION

The `FPReadExt` command retrieves a range of bytes from a specified fork. The server begins reading at the byte number specified by the offset field. 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 field.

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 `afpEOFError` or `LockErr`, respectively.

The Newline Mask is a byte mask that is to be logically ANDed with a copy of each byte read. If the result matches the Newline Char, the read terminates. Using a Newline Mask of $00 essentially disables the Newline check feature.

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

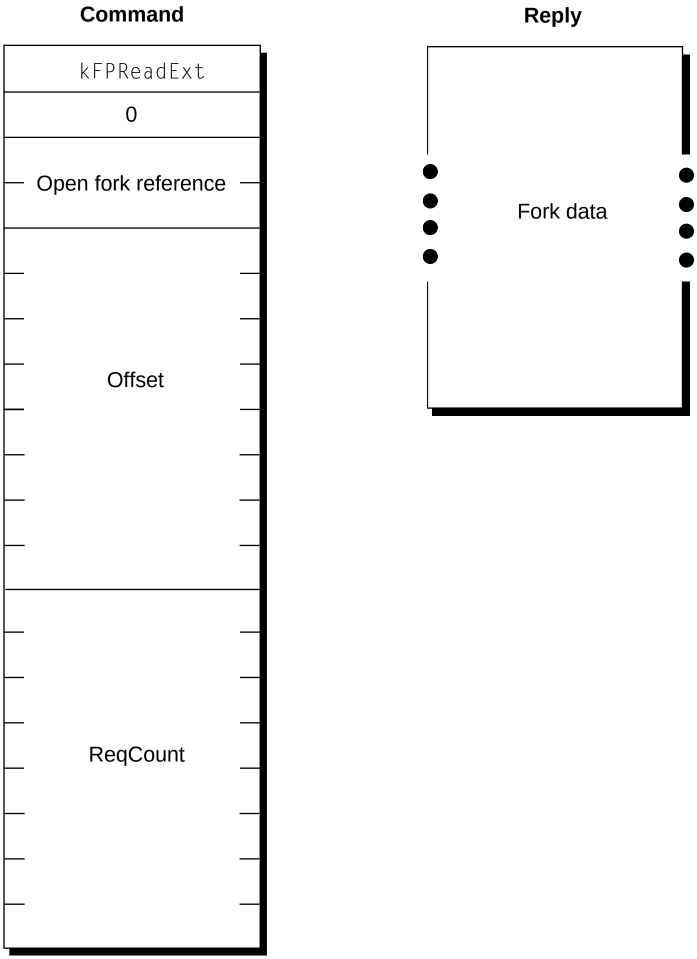
Lock the range to be read before issuing this command. The underlying transport mechanism may force the request to be broken into multiple smaller requests. If the range is not locked when this command begins execution, it is possible for another user to lock some or all of the range before this call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

The `ActCount` value is returned by the underlying transport mechanism and not as a parameter in the reply block.

PRIVILEGES

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

Figure 2-17 Command and reply blocks for the FPReadExt command



FPResolveID

Returns parameters for the file referred to by the specified file ID.

Inputs	*VolumeID* (int)	The ID of the volume on which the file ID is located.
	FileID (long)	The file ID that is to be resolved.
	ResultBitmap (int)	Bitmap describing which parameters are to be returned. (The bitmap structure is shown later in this section.)
Outputs	*ResultBitmap* (int)	Copy of input parameter.
	Requested parameters	
	FPError (long)	
Result codes	`afpCallNotSupported`	The AFP version is earlier than 2.1.
	`afpIDNotFound`	File ID was not found. (No file thread exists.)
	`afpObjectTypeErr`	Object defined was a directory, not a file.
	`afpBadIDErr`	File ID number is not a defined file ID.
	`afpAccessDenied`	User does not have the privileges required to issue this command.
	`afpParmErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

VERSION

Supported by AFP 2.1 and later.

DISCUSSION

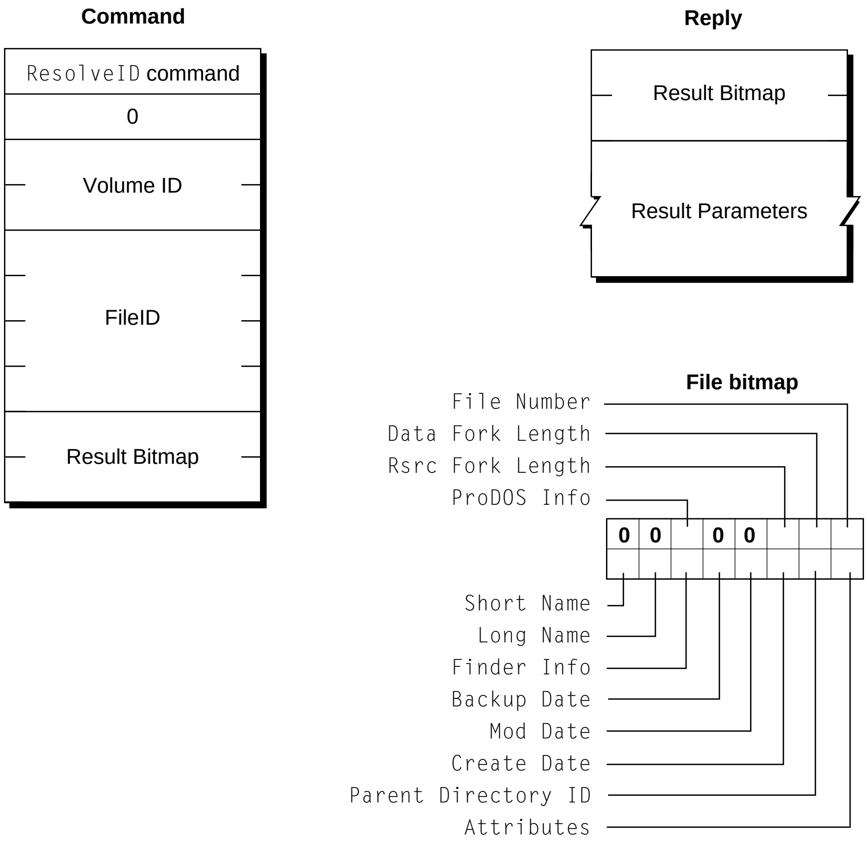
The `FPResolveID` command returns parameters for the file referred to by the specified file ID. The parameters can be any of those specified in the `FPGetFileDirParms` command: Short Name, Long Name, Finder Info, Backup

Date, Modification Date, Creation Date, Parent Directory ID, File Number, Data Fork Length, Resource Fork Length, and ProDOS Info.

Before issuing this command, the client must have called `FPOpenVol` for this volume.

Figure 2-18 shows the command and reply blocks for the `FPResolveID` command.

Figure 2-18 Command and reply blocks for the FPResolveID command



PRIVILEGES

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

FPWriteExt

Writes data write a block of data to an open fork.

Inputs	*pad* (int)	Pad byte.
	forkRef (int)	Open fork reference number.
	offset (16 bytes)	Byte offset from the beginning or end of the fork to where the write is to begin; a negative value indicates a byte within the fork relative to the end of the fork.
	reqCount (16 bytes)	Number of bytes to be written.
Outputs	*FPError* (long)	
	ActCount (16 bytes)	Number of bytes actually written to the fork.
	LastWritten (16 bytes)	Number of the byte just past the last byte written.
Result codes	`afpParmErr`	Session reference number or open fork reference number is unknown.
	`afpAccessDenied`	Fork was not opened for write access.
	`LockErr`	Some or all of the requested range is locked by another user.
	`DiskFull`	The volume is full.

VERSION

Supported in AFP 3.0 and later.

DISCUSSION

The `FPReadWrite` command writes a block of data to an open fork.

The Start/End flag allows a block of data to be written at an offset relative to the end of the fork. Therefore, data can be written to a fork when the user does not know the exact end of the fork, as can happen when multiple writers are concurrently modifying a fork. The server returns the number of the byte just past the last byte written.

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`. 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 LockErr result code and does not write any data to the fork.

The file's modification date is not changed until the fork is closed.

Lock the range before submitting 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 allof the range before this command completes, causing the write to success partially.

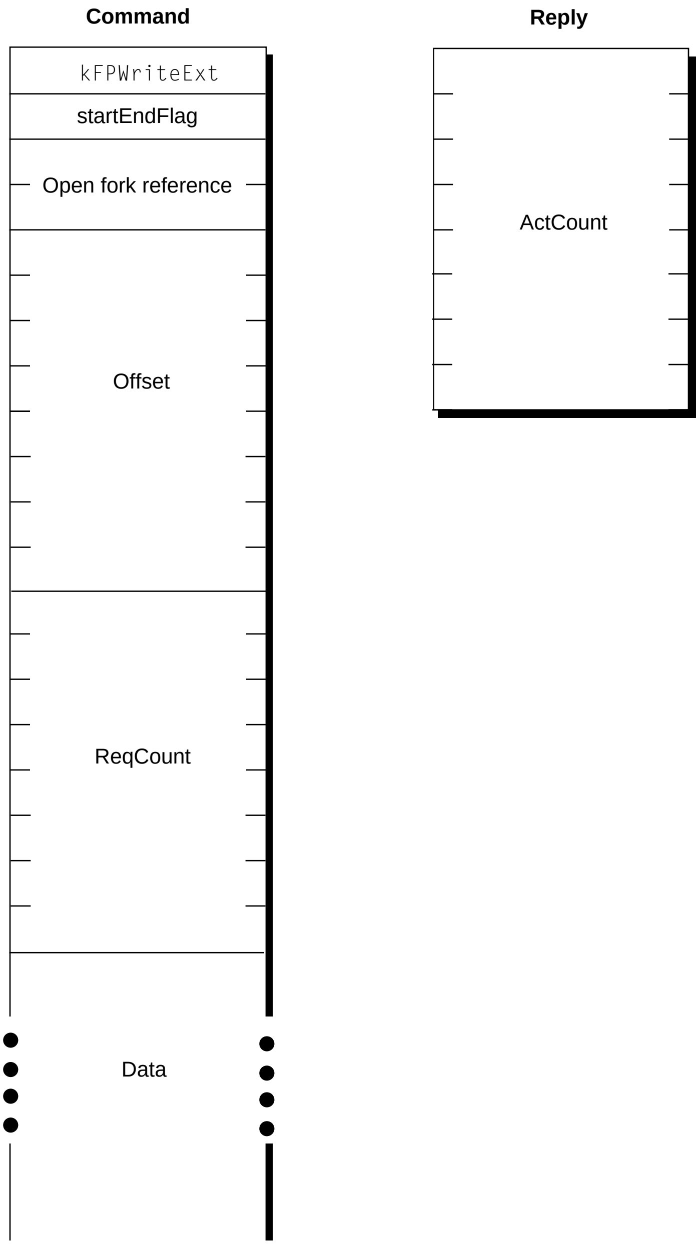
The data to be written is transmitted to the server in an intermediate exchange of ASP packets.

PRIVILEGES

The user must have the Read & Write privilege to issue this command.

Figure 2-18 shows the command and reply blocks for the `FPResolveID` command.

Figure 2-19 Command and reply blocks for the FPWriteExt command



Result Codes

Result Codes Added for AFP 2.1 and Later

Table 2-5 lists the additional result codes defined for AFP version 2.1 and later. Each result code is a 4-byte long word.

Table 2-5 Additional result codes defined for AFP version 2.1 and later

Constant	**Result Code**	**Description**
`afpIDNotFound`	–5034	Returned when the file ID was not found. (No file thread exists.)
`afpIDExists`	–5035	Returned when an attempt is made to create a file ID for a file that already has a file ID.
`afpCatalogChanged`	–5037	Returned when the catalog has changed while an `FPCatSearch` operation was being performed. *CatPosition* is not returned. The client must restart the search by setting the first word of *CatPosition* to zero.
`afpSameObjectErr`	–5038	Returned when an attempt is made to create a file ID for a file that already has a file ID.
`afpBadIDErr`	–5039	Returned when an `FPResolveID` operation is performed on a nonexistent file ID. (File ID is dangling or doesn't match the file number.)
`afpPwdSameErr`	–5040	Returned when the user attempts to change his or her password to the same password that he or she previously had.

Constant	Result Code	Description
`afpPwdTooShortErr`	–5041	Returned when the user's password is too short, or the user attempts to change his or her password to a password that is shorter than the server's minimum password length.
`afpPwdExpiredErr`	–5042	Returned when the user's password has expired and the user is required to change his or her password. The user can log on, but can only perform an `FPChangePassword` operation.
`afpInsideSharedErr`	–5043	The folder being shared is inside a shared folder; the folder contains a shared folder and is being moved into a shared folder; or the folder contains a shared folder and is being moved into the descendent of a shared folder. `FPMoveAndRename` may return this error.
`afpInsideTrashErr`	–5044	The folder being shared is inside the trash folder; the shared folder is being moved into the trash folder; or the folder is being moved to the trash and it contains a shared folder. `FPMoveAndRename` may return this error.

Result Codes Added for AFP 2.2 and Later

Table 2-6 lists the additional result code defined for AFP version 2.2 and later. The result code is a 4-byte long word.

Table 2-6 Additional result code defined for AFP version 2.2 and later

Constant	Result Code	Description
`afpPwdNeedsChangeErr`	–5045	Returned when the server requires the user to change his or her password before logging on.

Result Codes Added for AFP 3.0 and Later

Index