AppleShare 3.0 Developer's Kit

AppleTalk Filing Protocol Version 2.1

Apple Computer, Inc.

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

Overview of the extension in AFP 2.1

This document describes extensions to version 2.0 of the AppleTalk Filing Protocol (AFP), the version currently used in AppleShare 2.0.1 and documented in *Inside AppleTalk,* that support extra features in AFP servers and new calls that have been added to the hierarchical file system (HFS) for system software version 7.0. These protocol extensions are called *AFP 2.1*. The `AFPVersion` string for AFP 2.1 is `AFPVersion 2.1`.

The following calls have been added to the protocol:

- `afpGetSrvrMsg`, which enables an AFP client to get a string message from the server. Use of this call is optional; the server can be considered AFP 2.1–compliant whether or not this call is supported. This document defines the previously undocumented `AFPUserBytes` field, the 2-byte attention code sent in an ASP Attention packet to an AFP client.
- `afpCreateID`, `afpDeleteID`, `afpResolveID`, and `afpExchangeFiles`, which support file IDs. File IDs provide a mechanism by which applications and users can keep track of a file regardless of whether it has been moved or its name has been changed. Use of these calls is optional. For more information, see "Volume Attributes Word in `afpGetVolParms`," later in this document.
- `afpCatSearch`, which allows searching of the catalog on almost any field that is returned by `PBGetCatInfo`. Use of this call is optional. For more information, see "Volume Attributes Word in `afpGetVolParms`," later in this document.

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 and HFS, the bitmaps of certain calls have been augmented:

- new Directory Attributes and Access Rights in `afpGetFlDrParms` and any call that uses this bitmap
- new Flags word bit definitions returned by `afpGetSInfo`
- new Volume Attributes in `afpGetVolParms`

A new user authentication method (UAM), known as *Two-Way Scrambled*, is available for use 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 to accommodate an environment 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 World privileges are copied to the new folder.

User and group names are now 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.

The rest of this document describes these extensions in more detail. The section "Some AFP 2.1–Related Questions and Answers," later in this document, provides additional information about AFP 2.1.

Blank access privileges

AFP 2.1 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.

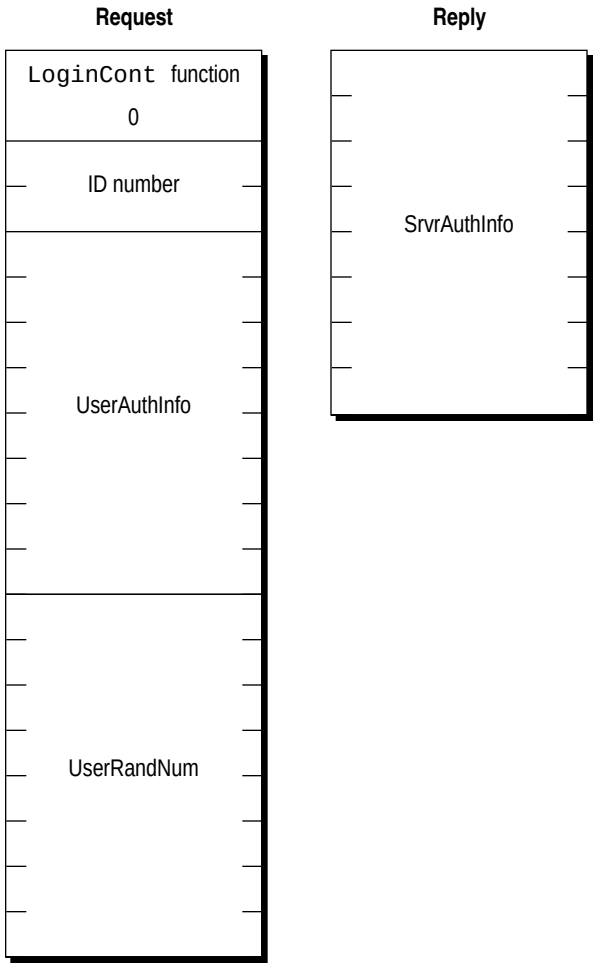
Two-Way Scrambled user authentication method

AFP 2.1 supports a new user authentication method, the Two-Way Scrambled 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 last step. The corresponding UAM string is `2-Way Randnum exchange`.

Both the Random Number Exchange UAM and the Two-Way Scrambled UAM start with the workstation asking to login to the server. If the login is allowed, the server returns a random double-long word and an error of `afpAuthContinue`. The workstation then
encodes the double-long word with its password and sends it back to the server in an `afpLoginCont` call. If the encoding was performed correctly, the workstation is authenticated and `noErr` is returned. However, for the Two-Way Scrambled method, the workstation sends a second random double-long word along with its `afpLoginCont` call. The server encodes this double-long word with what it believes is the user's password, and returns the resulting double-long word in the `afpLoginCont` reply.

The workstation compares this response to what resulted from its encoding of the second double-long word; if the two responses are the same, the server is then also authenticated. This feature guards against trojan-horse file servers.

The following figure shows the command and reply block formats for the `afpLoginCont` call when the Two-Way Scrambled user authentication method is used.



The Two-Way Scrambled UAM is not available for use with the `afpPwdChange` call, nor is it required. If the user is concerned about authenticating the server, he or she will have already logged in to the server with the Two-Way Scrambled UAM. Since the user must already be authenticated to call `afpPwdChange`, 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 Scrambled UAM use 8-bit ASCII characters in the password. 7-bit ASCII is used only by the Cleartext UAM.

The Random Number Exchange and Two-Way Scrambled UAMs interpret differently the password used as the key passed to the National Bureau of Standards Data Encryption Standard (NBS DES) algorithm.

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 NBS 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 Scrambled UAM, the key is shifted left one bit (that is, with the 68000 LSL instruction) 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.)

New bitmap definitions

This section describes the new bits defined for AFP 2.1. The bits are divided into three categories: Directory Attributes and Access Rights words in `afpGetFlDrParms`, Flags word in `afpGetSInfo`, and Volume Attributes word in `afpGetVolParms`.

Directory Attributes and Access Rights words in `afpGetFlDrParms`

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), new bit definitions have been added to the Directory Attributes word for `afpGetFlDrParms`:

`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 a superuser can see or modify).

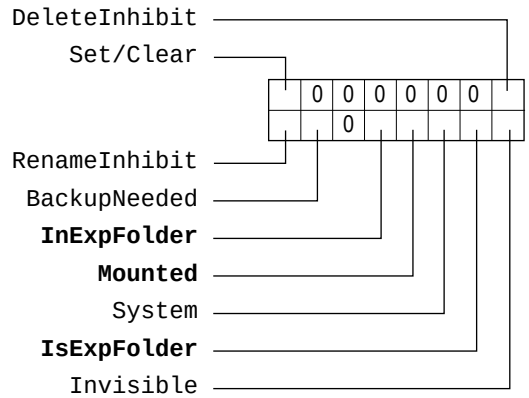
`Mounted` (bit 3)
This share point is mounted by a regular user (that is, a user without “All Privileges”). The icon for such a folder indicates to the user of the local computer, that this folder is a share point, and that a remote user currently has it mounted.

`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 `afpSetFileDirParms`. They are returned to the remote user and are relevant to a general AFP server since 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).

The following figure shows the entire Directory Attributes word, with the new bits for AFP 2.1 shown in boldface.

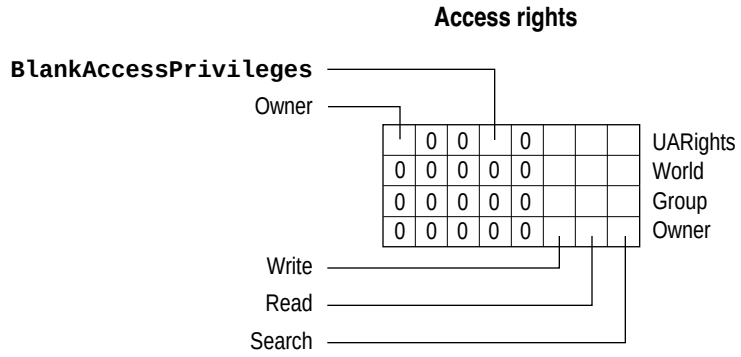
Directory attributes



To accommodate blank access privileges, a new bit definition has been added to the Access Rights long word for `afpGetFlDrParms`:

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

The following figure shows the entire Access Rights long word, with the new bit for AFP 2.1 in boldface.



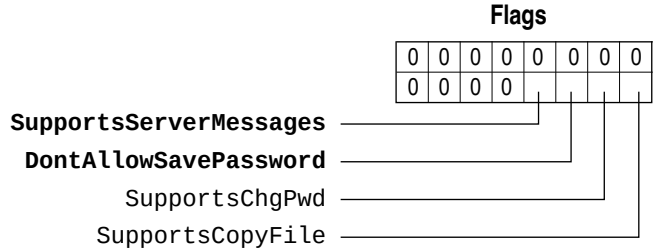
Flags word in `afpGetSInfo`

In order to accommodate the (optional) new features of AFP 2.1, some new bit definitions have been added to the Flags word for `afpGetSInfo`. The new bits are as follows:

`DontAllowSavePassword` (bit 2)
The workstation 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, but when this bit is set, the button that would allow the user to save his or her name and password will not be displayed.

`SupportsServerMessages` (bit 3)
Since server messages are an option in AFP 2.1, this bit allows servers to specify whether or not this optional feature is supported.

The following figure shows the entire Flags word, with the new bits for AFP 2.1 in boldface.



Volume Attributes word in `afpGetVolParms`

In order to accommodate the new HFS calls in System 7, some new bit definitions have been added to the Volume Attributes word for `afpGetVolParms`:

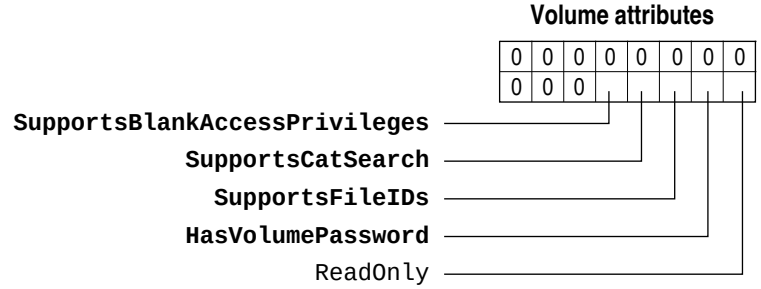
`HasVolumePassword` (bit 1)
This volume has a volume password. Volume passwords were supported in prior versions of AFP; now the volume attributes reflect this information. This bit has the same value as the `HasPassword` bit returned for each volume by `afpGetSrvrParms`.

`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.

`SupportsCatSearch` (bit 3)
This volume supports `afpCatSearch` calls. Since the use of `afpCatSearch` 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. Blank access privileges are discussed earlier in this document.

The following figure shows the entire Volume Attributes word, with the new bits for AFP 2.1 in boldface.



New security features

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

Minimum password length

It is now possible to 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 `afpPwdTooShort` error upon logging in. The client code should display an explanatory dialog box and then allow the user to change his or her password. The `afpPwdChange` call will continue to fail with an `afpPwdTooShort` 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 login 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

It is now possible to 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 `afpPwdChange` call and change the password; issue an `afpLogout` call; or issue an `afpLoginCont` call. (If the user issues any other call, the error `afpParmErr` will be returned.) The `afpLoginCont` call returns one of the following errors: `afpPwdTooShortErr`, `afpPwdExpiredErr`, or `afpPwdNeedsChangeErr`. At this point the user is logged in, and the only command that can be issued is `afpPwdChange` or `afpLogout`. If the user issues any other command, the error `afpParmErr` will be 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, `afpGetSParms` and `afpOpenVol`, will allow the user to log in 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 will no longer be able to connect to the server.

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

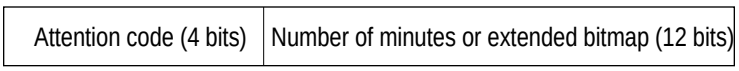
Maximum failed login attempts

It is now possible to specify the maximum number of consecutive failed login 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 login. For every failed login attempt without a preceding successful login, the count is incremented. When the maximum number of failed login attempts is reached, the user's account is disabled. Any attempts to log in after the account is disabled will result in an `afpParmErr` indicating that the user is unknown or that his or her login is disabled. The administrator will need to be notified to 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.

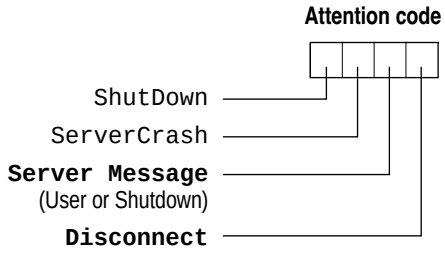
New `AFPUserBytes` definitions

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 have been augmented to accommodate some of the new features in AFP 2.1 (such as the server message feature) and new modes in the workstation code (such as `Disconnect`).

The `AFPUserBytes` bytes are defined as shown here.



The following figure shows how the attention code bits for the `AFPUserBytes` bytes are defined, with the new bit definitions for AFP 2.1 in boldface.



The bit numbers for the attention code bits are defined as follows:

bit 15 ShutDown bit. This bit is used when the session is being shut down. Either the server is being shut down or the user is being disconnected.

bit 14 ServerCrash bit. The server has detected some 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.

bit 13 Server Message bit. There is a server message that the client should request by using the `afpGetSrvrMsg` call with a *MsgType* of "Server." (For more information, see the section "`afpGetSrvrMsg` (38 or $26)" later in this document.) The client should request the message as soon as possible after receiving this attention code, or else the server message it receives could be outdated.

bit 12 Disconnect bit. This bit is set when the user is being disconnected. This bit has meaning only if the ShutDown bit is also set.

Here is a list of valid bit combinations:

1000 The server is shutting down in the designated number of minutes. No message accompanies this shutdown. This code may be used upon server shutdown (that is, when the administrator quits file service).

1001 The user will be disconnected in the designated number of minutes. No message accompanies this shutdown. This is one of the codes used upon user disconnection (for example, when the administrator detects an intruder and disconnects him or her).

1010 The server is shutting down in the designated number of minutes. A message accompanies this shutdown. The workstation should immediately submit an afpGetSrvrMsg call to receive and display the message. This code may be used upon server shutdown (that is, when the administrator quits file service).

1011 The user will be disconnected in the designated number of minutes. A message accompanies this shutdown. The workstation should immediately submit an afpGetSrvrMsg call 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.

0010 The server has a server message available for this workstation. The workstation should immediately submit an afpGetSrvrMsg call to receive and display the message. The extended bitmap is reserved for Apple Computer's use only.

0011 Reserved. The extended bitmap is reserved for Apple Computer's use only.

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 twelve 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.

`afpGetSrvrMsg` (38 or $26)

The `afpGetSrvrMsg` call allows an AFP client to get a string message from the server. This call is made by the client to receive shutdown, user, and login messages from the server. Usually, the server will send an attention code to the client when these messages are available. However, the client can make the `afpGetSrvrMsg` call at any time. An empty or zero-length string is returned if no message is available.

The `afpGetSrvrMsg` parameters are defined as follows:

Inputs

MsgType (int)	Type of server message: 0 = `login` 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 = login 1 = server
MsgBitmap (int)	Bitmap indicating what information was passed.
SrvrMessage (str)	String message from the server.
FPError (long)	

Result codes

`afpCallNotSupported`	`afpGetSrvrMsg` is not implemented by the server, or the AFP version is earlier than 2.1.
`afpUserNotAuth`	The user was not logged in.
`afpBitmapError`	The bitmap specified has unrecognized bits set.

Rights

The client must be logged on to the server to receive server message notification and to issue this request. Other than that, the client need have no special access rights to issue this call.

Continued on following page ▶

The `login` (0) *MsgType* is used for only one kind of message:

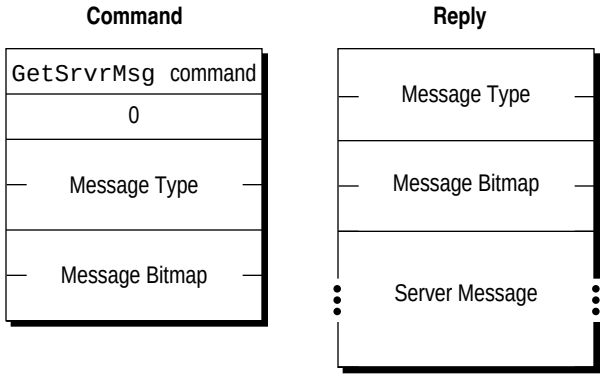
Login This condition allows the server to send a message to a client at login time. The workstation can query the server for a login message at login time, or whenever it is convenient to do so. If there is no login message, `afpGetSrvrMsg` returns a zero-length string, and nothing need be displayed.

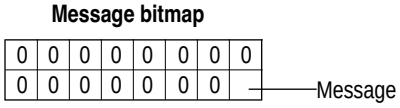
The `server` (1) *MsgType* is used for two kinds of messages:

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 workstation 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 workstation is made aware that a user message is available by the server's setting the Server Message bit in `AFPUserBytes`. Workstations implementing older AFP versions should simply 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 workstation know that there is a server message. The client then requests (by means of `afpGetSrvrMsg`) the message from the server.





`afpCreateID` (39 or $27)

As stated earlier, file IDs provide a means of keeping track of a file even if its name or location changes. The `afpCreateID` call creates a unique file ID for a specified file. (Note that the scope of file IDs is limited to the files on a volume. File IDs cannot be used across volumes.)

The `afpCreateID` parameters are defined as follows:

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 ID is to be created.
PathType (byte)	Path type of the pathname: 1 = short name 2 = long name
PathName (str)	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

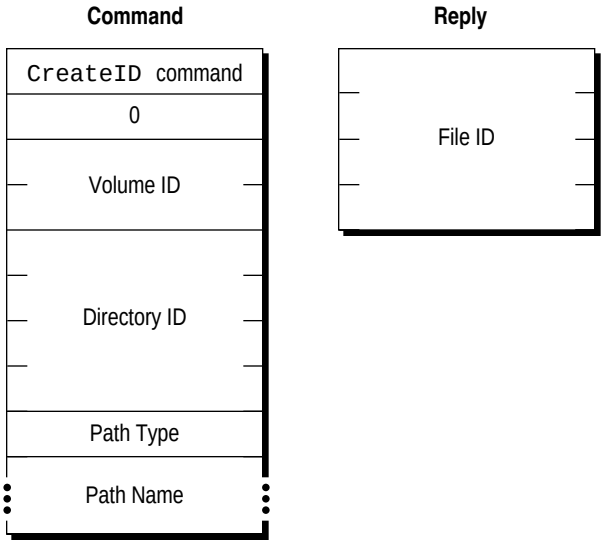
`afpCallNotSupported`	AFP version earlier than 2.1.
`afpObjectNotFound`	The target file does not exist.
`afpIDExists`	A file ID already exists for this file. The file ID is returned in the `FileID` field.
`afpObjectTypeErr`	Object defined was a directory, not a file.
`afpVolLocked`	The destination volume is read-only.
`afpAccessDenied`	User does not have the rights required to issue this call.
`afpParamErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

Continued on following page ▶

Rights The user must have See Files rights to issue this call.

Notes Before issuing this call, the user must have called `afpOpenVol` 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.



`afpDeleteID` (40 or $28)

The `afpDeleteID` call invalidates all instances of the specified file ID.

The `afpDeleteID` parameters are defined as follows:

Inputs

VolumeID (int)	The ID of the volume on which the file ID is to be invalidated.
FileID (long)	File ID that is to be invalidated.

Outputs

FPError (long)

Result codes

`afpCallNotSupported`	AFP version 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 rights required to issue this call.
`afpParamErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

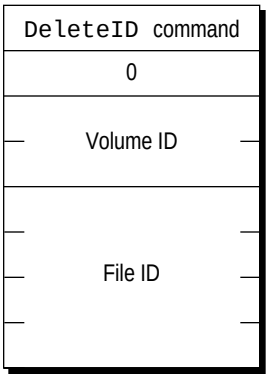
Rights

The user must have See Files and Make Changes rights to issue this call.

Notes

Before issuing this call, the user must have called `afpOpenVol` for this volume.

Command



`afpResolveID` (41 or $29)

The `afpResolveID` call returns parameters for the file referred to by the specified file ID. These parameters can be any of those specified in the `afpGetFlDrParms` call.

The `afpResolveID` parameters are defined as follows:

Inputs

VolumeID (int)	The ID of the volume on which the file ID is located.
FileID (long)	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

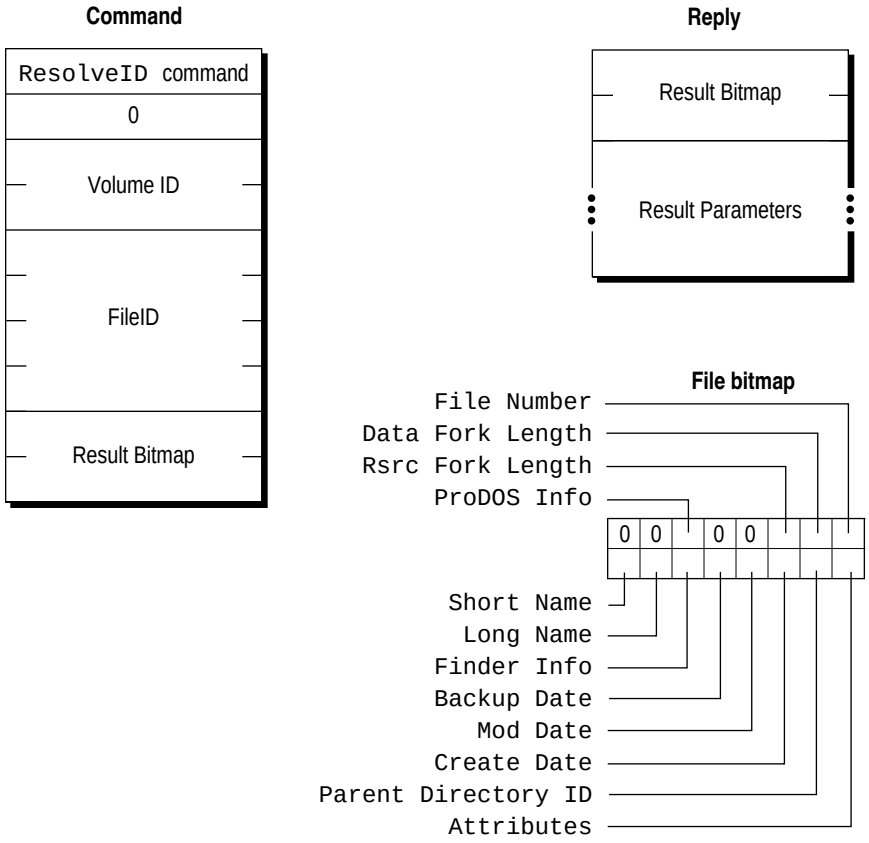
ResultBitmap (int)	Copy of input parameter.
Parameters requested	
FPError (long)	

Result codes

`afpCallNotSupported`	AFP version 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 rights required to issue this call.
`afpParamErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

Rights The user must have See Files rights to issue this call.

Notes Before issuing this call, the user must have called `afpOpenVol` for this volume.



`afpExchangeFiles` (42 or $2A)

The `afpExchangeFiles` call is used to preserve existing file IDs when an application performs “Save” or “Save As” functions. Both files to be changed are specified. They must exist on the same volume. File IDs do not have to exist on the files to be exchanged. The files can be either open or closed.

The `afpExchangeFiles` parameters are defined as follows:

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 (str)	String name of the source file.
DestPathType (byte)	Path type of the destination pathname: 1 = short name 2 = long name
DestPathName (str)	String name of the destination file.

Outputs

FPError (long)

Result codes

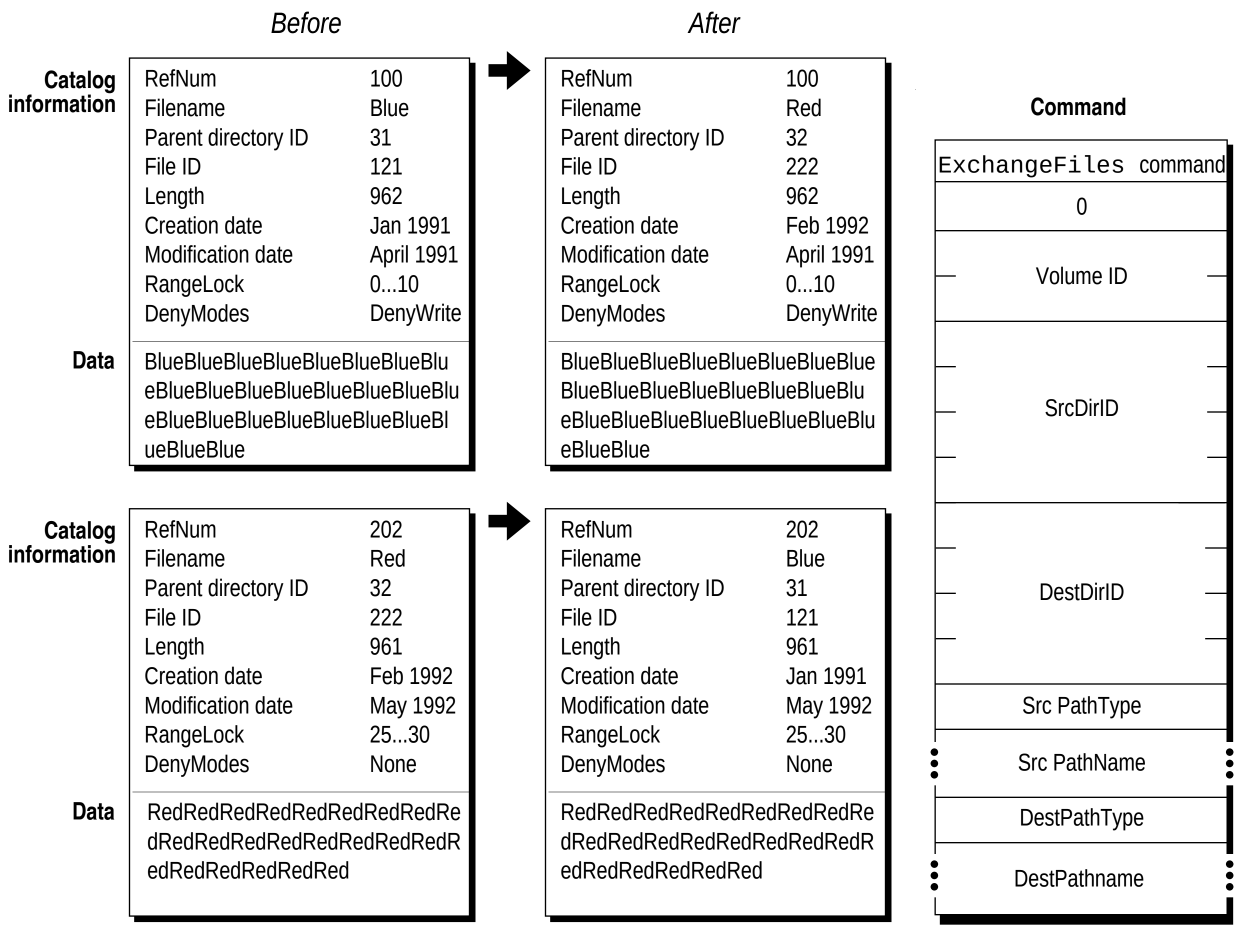
`afpCallNotSupported`	AFP version earlier than 2.1.
`afpObjectNotFound`	A target file does not exist.
`afpObjectTypeErr`	Object defined was a directory, not a file.
`afpObjectLocked`	The file was locked.
`afpSameObjectErr`	The source file is the same as the destination file.
`afpVolLocked`	The destination volume is read-only.
`afpAccessDenied`	User does not have the rights required to issue this call.
`afpParamErr`	Session reference number, volume identifier, or pathname type is unknown; pathname is null or bad.

Rights

The user must have See Files and Make Changes rights to both files to issue this call.

Notes Before issuing this call, the user must have called `afpOpenVol` for this volume.

The following example shows the results of an `afpExchangeFiles` operation between the two files “Blue” and “Red.” 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.



`afpCatSearch` (43 or $2B)

The `afpCatSearch` call allows an application to efficiently search an entire volume for files that match specified criteria. These criteria include any fields in the file bitmaps, directory bitmaps, or both, that are defined for the `afpGetFlDrParms` call. Information parameters for the matching files and directories are returned. These parameters can also be any of those specified for the `afpGetFlDrParms` call.

The `afpCatSearch` call's parameters are defined as follows:

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 parameters that are to be returned; this field is the same as the `File Bitmap` field in the `afpGetFlDrParms` call (with some restrictions, explained later in this section).
DirRsltBitmap (int)	The fields in the `Dir` parameters that are to be returned; this field is the same as the `Directory Bitmap` field in the `afpGetFlDrParms` call (with some restrictions, explained later in this section).
RequestBitmap (long)	The fields in the `File/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 that describes the matches that were found.
FPError (long)	

Result codes

`afpCallNotSupported`	AFP version earlier than 2.1.
`afpCatalogChanged`	The catalog has changed and *CatPosition* may be invalid. No matches are returned.
`afpParmErr`	Input parameters are not valid or volume identifier is unknown.
`afpEofError`	No more matches.

Rights

The user need have no special access rights to issue this call; however, to see all the files, folders, or files and folders that match the specified criteria, the caller must have See Files/See Folders rights to them. Folders for which the caller does not have See Files/See Folders rights are skipped by the search.

Notes

Before issuing this call, the user must have called `afpOpenVol` for this volume.

CatPosition is a 16-byte field in which the first word signifies whether the field denotes a "real" catalog position or hint. If the first word is zero, `afpCatSearch` should start from the beginning of the volume. If the first word is nonzero, *CatPosition* is a "real" catalog position and `afpCatSearch` should begin its search with this entry.

Specification1 and *Specification2* are used together to specify the search parameters. These parameters are packed in the same order as that in which the bits are set in the request bitmap. All variable-length parameters (*name,* for example) are put at the end of each spec-ification 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.

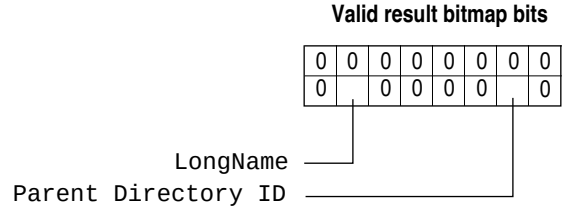
Continued on following page ▶

The `afpCatSearch` call returns the error `afpEofError` only when it has reached the end of the volume directory tree. For example, if the workstation requests ten matches, the server may return only four matches, without returning an error. The workstation 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 the `afpCatSearch` call returns the error `afpCatalogChanged`, the workstation cannot continue the search. The workstation must restart the search by setting the first word of *CatPosition* to zero.

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

Valid bitmaps for `afpCatSearch`

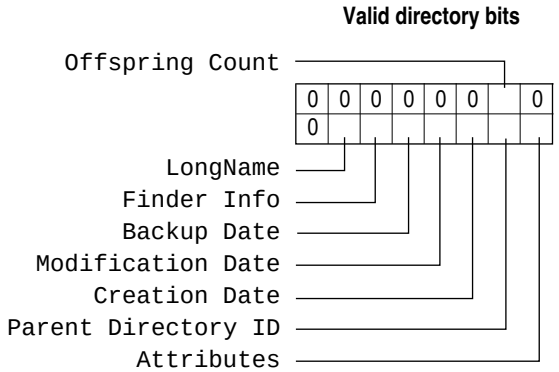
The only valid bits for the `FileRsltBitmap` and `DirRsltBitmap` fields are the `LongName` and `Parent Directory ID` bits. The following figure shows the valid Result Bitmap bits.



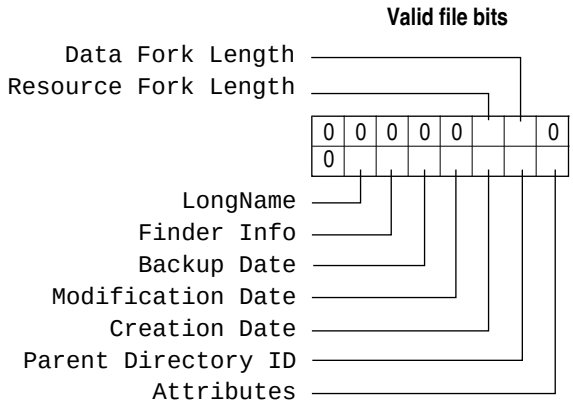
RequestBitmap

The low-order word of *RequestBitmap* is roughly equivalent to the File and Directory Bitmaps in `afpGetFlDrParms`. (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 name (0 = full name, 1 = partial name). There is no equivalent to the `fsSBNegate` bit used by the Macintosh File Manager's `PBCatSearch` function.

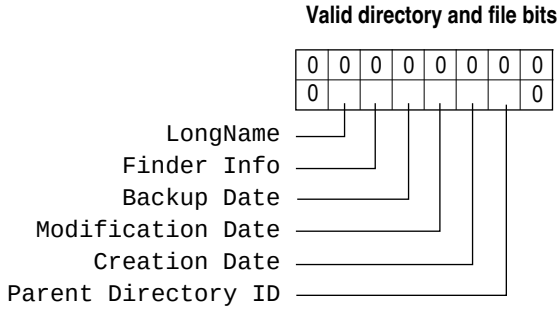
The following figure shows the valid directory bits. `afpCatSearch` can search for this information when searching for directories only.



The following figure shows the valid file bits. `afpCatSearch` can search for this information when searching for files only.

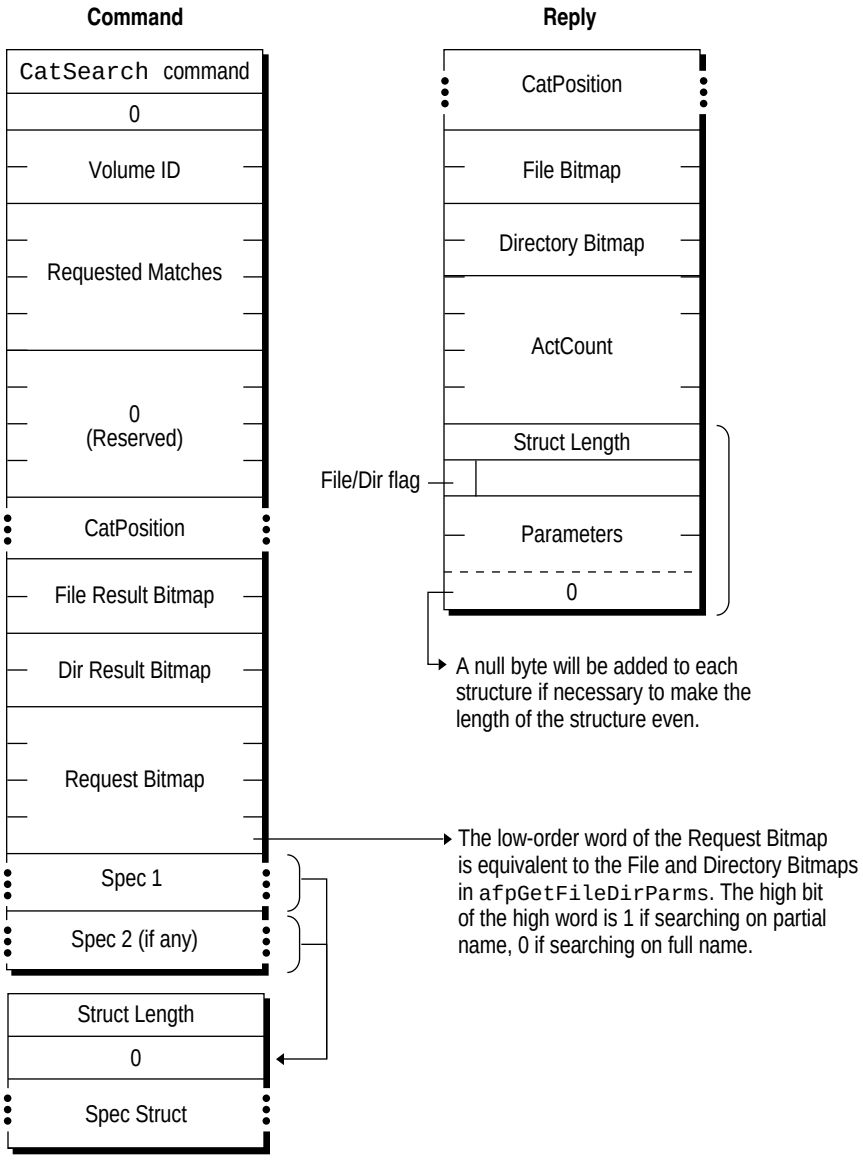


The following figure shows the valid directory and file bits. `afpCatSearch` can search for this information when searching for directories and files.



Attributes bits

The only valid bits that can be searched for in the *Attributes* parameter are the inhibit bits. 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 searching for files and directories.



New function codes

The following new function codes have been defined for AFP 2.1. Each function code is a 16-bit integer sent high-byte first in the packet.

Decimal value	*Hex value*	*AFP function*
38	$0026	`afpGetSrvrMsg`
39	$0027	`afpCreateID`
40	$0028	`afpDeleteID`
41	$0029	`afpResolveID`
42	$002A	`afpExchangeFiles`
43	$002B	`afpCatSearch`

New result codes

The following new result codes have been defined for AFP 2.1. Each result code is a 4-byte long word.

Decimal	Hex	FPError	Description
–5034	$FFFFEC56	`afpIDNotFound`	Returned when the file ID was not found. (No file thread exists.)
–5035	$FFFFEC55	`afpIDExists`	Returned when an attempt is made to create a file ID for a file that already has a file ID.
–5037	$FFFFEC53	`afpCatalogChanged`	Returned when the catalog has changed while an `afpCatSearch` operation was being performed. *CatPosition* is not returned. The workstation must restart the search by setting the first word of *CatPosition* to zero.
–5038	$FFFFEC52	`afpSameObjectErr`	Returned when `afpExchangeFiles` is called and the source and destination files are the same.
–5039	$FFFFEC51	`afpBadIDErr`	Returned when an `afpResolveID` operation is performed on a nonexistent file ID. (File ID is dangling or doesn't match the file number.)
–5040	$FFFFEC50	`afpPwdSameErr`	Returned when the user attempts to change his or her password to the same password that he or she previously had.
–5041	$FFFFEC4F	`afpPwdTooShort`	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.
–5042	$FFFFEC4E	`afpPwdExpired`	Returned when the user's password has expired and the user is required to change his or her password. The user can log in, but can only perform an `afpPwdChange` operation.
–5043	$FFFFEC4D	`afpInsideSharedErr`	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. `afpMoveAndRename` may return this error.
–5044	$FFFFEC4C	`afpInsideTrashErr`	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. `afpMoveAndRename` may return this error.

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

Note that the AFP 2.1 server will respond 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 `afpGetFlDrParm`?

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 `afpResolveID` to see if 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 (in which case new values for *DTRefNum* and *Volume ID* are assigned).

Is it true that `afpCloseVol` 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 `afpCloseVol` to close them for you.