Apple Filing Protocol Version 3.1

October 16, 2003

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 document is accurate. Apple is not responsible for typographical errors.

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

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

Figures, Listings, and Tables

Chapter 1 Concepts 13

Chapter 3 Apple Filing Protocol Reference 79

About This Manual

The Apple Filing Protocol (AFP) allows users of multiple computers to share files easily and efficiently over a network.

What's New

The following changes have been made to AFP since AFP version 3.0:

- A new command, FPEnumerateExt2, has been added. It is intended to be used instead of the FPEnumerateExt command. The StartIndex and MaxReplySize components of the FPEnumerateExt2 request block are longs (instead of shorts), which allows more entries in a single directory to be listed.
- Improved support for recovering from an AFP client crash. Two new parameters have been added to the request block for the `FPGetSessionToken` command, and new values have been defined for the `Type` parameter. The new parameters allow an AFP client to associate a unique value with a session. If the local system crashes while the AFP client has files open on a remote server, the AFP client can log in again and send the `FPGetSessionToken` command to tell the server to close files that were open so that they can be opened again by the new session.
- The meaning of the ServerCrash bit in the Attention code of AFPUserBytes has changed. If the ServerCrash bit is set to 1, reconnect is allowed. If the ServerCrash bit is set to 0, reconnect is not allowed.
- Two new user authentication methods (UAMs), Kerberos and Diffie-Hellman Exchange 2 (DHX2) are available. The Kerberos UAM is an implementation of Kerberos from MIT. DHX2 is similar to the Diffie-Hellman Exchange, but it uses a variable length prime to provide scalable security and allows the use of longer passwords.
- Support has been added for disabling traditional mapping of User ID and Group ID permissions when the AFP client is a UNIX client connected to a Mac OS X Server. Instead of showing the traditional mappings, the AFP client can display the actual UNIX user ID, group ID, and permissions. A new bit, `kNoNetworkUserIDs`, is defined in a volume's Attributes parameter to indicate whether the AFP client is to perform traditional mapping or display UNIX user IDs, group IDs, and permissions.
- A new bit (UTF-8 ServerName) is defined in the `Flags` word returned by the `FPGetSrvrInfo` command and, if set, `FPGetSrvrInfo` returns a new field in the reply block. Bit 9 in the `Flags` word indicates that the server provides a UTF-8 version of the server name. If it is set, the

next two bytes after the DirectoryNamesCount Offset is an offset to the UTF-8 name. The name is an AFP name.

- A timestamp parameter has been added to the request block for the `FPGetSessionToken` command, and new values have been defined for the `Type` parameter. Two previously defined values for the `Type` parameter, `kLoginWithID` and `kReconnWithID`, have been deprecated.
- A new UAM, the Reconnect UAM, has been added to support reconnections after a server reboot.
- New reconnect HI notifies the user when a server is not responding and allows the user to disconnect from the server.
- A second timer has been added to the existing session maintenance timer for AFP/TCP in order to improve the accuracy of determining when a disconnect has occurred.
- Support for IPv6 has been added. The server now provides information about the server's support for IPv6 in the Network Address field returned by the `FPGetSrvrInfo` command.
- A new value, `kGetKerberosSessionKey` for the `Type` parameter of the `FPGetSessionToken` command has been added. This value is used by clients that authenticate using Kerberos v5 in order to get a Kerberos v5 session key.

Conventions Used in This Manual

The `Letter Gothic` font is used to indicate text that you type or see displayed. This manual 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 sources provide additional information that may be of interest to developers who use the Open Directory programming interface:

- *Inside AppleTalk*. Addison Wesley. ISBN 0-201-19257-8
- Diffie-Hellman Key Agreement Applied Cryptography Chapter 22
- CAST 128, RFC 2144
- SSLeay/OpenSSL, available from `http://www.openssl.org`

Concepts

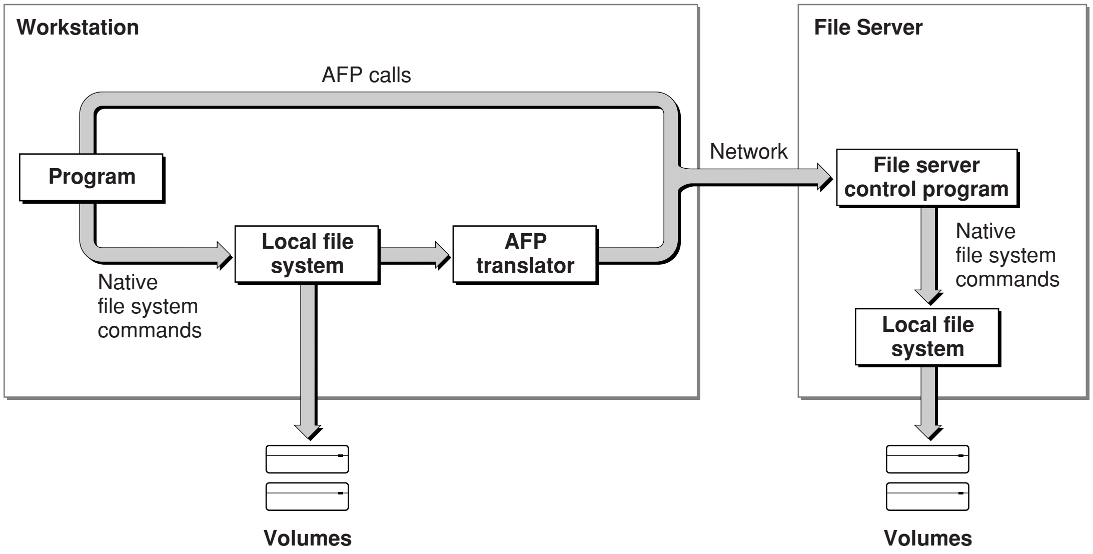
The Apple Filing Protocol (AFP) allows users of multiple computers to share files easily and efficiently over a network.

File Access Model

This section introduces the file access model used by AFP to enable file sharing and discusses the components of AFP software.

Note: All values exchanged between an AFP client and an AFP server are sent over the network in network byte order.

Figure 1-1 AFP file access model



A program running in a local computer requests and manipulates files by using that computer's native file system commands. These commands manipulate files on disk or other memory resource that is physically connected to the local computer. Through AFP, a program can use the same native file system commands to manipulate files on a shared resource that resides on a remote computer (for example, a file server).

A program running on the local computer sends a command to the native file system. A data structure in local memory indicates whether the volume is managed by the native file system or by some external file system. The native file system discovers whether the requested file resides locally or remotely by looking at this data structure. If the data structure indicates an external file system, the native file system routes the command to the AFP translator.

The translator, as its name implies, translates the native commands into AFP commands and sends them through to the file server that manages the remote resource. The AFP translator is not defined in the AFP specification; it is up to the applications programmer to design it.

A program running on the local computer may also need to send AFP commands for which no equivalent command exists in the native file system. In this case, the AFP command is sent directly to the desired external file system, as shown in Figure 1-1. For example, user authentication might have to be handled through an interface written for that purpose.

AFP supports computers using Mac OS and personal computers using MS-DOS. AFP can be extended to support additional types of computers. Any implementation of AFP must take into account the capabilities of the networked computer's native file system and simulate its functionality in the shared environment. In other words, the shared file system should duplicate the characteristics of a local computer's file system. Simulating the functionality of each local computer's native file system becomes increasingly complex as different computer types share the same file server. Because each computer type has different characteristics in the way it manipulates files, the shared file system needs to possess the combined capabilities of all computers on the same network.

Three system components make up AFP:

- a file system structure, which is made up of resources (such as file servers, volumes, directories, files, and forks) that are addressable through the network. These resources are called AFP-file-system-visible entities. AFP specifies the relationship between these entities. For example, one directory can be the parent of another. For descriptions of AFP-file-system-visible entities, see "File System Structure" later in this chapter.

- AFP commands, which are the commands the local computer uses to manipulate the AFP file system structure. As mentioned earlier, the translator sends file system commands to the file server in the form of AFP commands, or the application running on the local computer can make AFP commands directly. Each AFP command is described in detail in the "Tasks" section of this document.

- algorithms associated with the commands, which specify the actions performed by the AFP commands.

Although this chapter distinguishes between local computers and file servers, AFP can support these two functions within the same node. However, AFP does not solve the concurrency problems that can arise when a computer is both an AFP client and an AFP server. The software on such combined nodes must be carefully designed to avoid potential conflicts.

AFP does not provide commands that support administration of the file server. Administrative functions, such as registering users and changing passwords, must be handled by separate network-administration software. Additional software must also be provided to add, remove, and find servers within the network.

File System Structure

This section describes the AFP file system structure and the parameters associated with its AFP-file-system-visible entities. These entities include the file server, its volumes, directories ("folders" in Mac OS terminology), files. and file forks. This section also describes the tree structure, called the volume catalog, which is a description of the relationships between directories and files.

By sending AFP commands, the AFP client can

- obtain information about the file server and other parts of the file system structure
- modify information that is obtained from a file server
- create and delete files and directories
- retrieve and store information within individual files

The following sections describe the file system structure's AFP-file-system-visible entities.

File Server

A file server is a computer with at least one large-capacity disk that allows other computers on the network to share information stored in it. AFP imposes no limit on the number of shared disks. Each disk attached to a file server usually contains one volume, although the disk may be subdivided into multiple volumes. Each volume appears as a separate entity to the AFP client.

A file server has a unique name and other identifying parameters. These parameters identify the server's machine type and number of attached volumes, as well as the AFP versions user authentication methods (UAMs) that the server supports. AFP file server parameters are listed Table 1-1.

Table 1-1 File server parameters

Parameter	Description
Server name	A string in Pascal format of up to 32 characters.
Server machine type	A string in Pascal format of up to 16 characters that describes the file server's hardware and software but has no significance to AFP.
Number of volumes	A two-byte integer.

Parameter	Description
AFP version strings	One or more strings of up to 16 characters each. For more information, see Table 1-12 (page 36).
UAM strings	One or more strings of up to 16 characters each. For more information, see Table 1-13 (page 36).
Server icon	A optional value of 256 bytes that is used to customize the appearance of server volumes on the Mac OS Desktop. It consists of a 32-by-32 bit (128 bytes) icon bitmap followed by a 32-by-32 bit (128 bytes) icon mask. The mask usually consists of the icon's outline filled with black (bits that are set). For more information about icons, refer to *Inside Mac OS X*.
Server signature	A 16-byte value that uniquely identifies a server used to prevent an AFP client from logging on to the same server twice.

Volumes

A file server can make one or more volumes visible to AFP clients. Each volume has parameters associated with it, as listed in Table 1-2 (page 16). To provide security at the volume level, the server can maintain an optional password parameter for any volume.

Table 1-2 Volume parameters

Constant and bit position	Parameter size	Description
`kFPVolAttributeBit` (bit 0)	Short	Volume attributes. See Table 1-3 (page 18) for details.
`kFPVolSignatureBit` (bit 1)	Two bytes	The volume signature identifies the volume type (flat, fixed Directory ID, or variable Directory ID). For details, see the section "Volume Types" (page 19).
`kFPVolCreateDateBit` (bit 2)	Four bytes	The date the server created the volume. This parameter cannot be modified by an AFP client.
`kFPVolModDateBit` (bit 3)	Four bytes	Updated by the server each time anything on the volume is modified. This parameter cannot be modified by an AFP client.
`kFPVolBackupDateBit` (bit 4)	Four bytes	Set by a backup program each time the volume's contents are backed up. When

		a volume is created, the Backup Date is set to 0x80000000 (the earliest representable date-time value).

Constant and bit position	Parameter size	Description
`kFPVolIDBit` (bit 5)	Two bytes	For each session between the server and an AFP client, the server assigns a Volume ID to each of its volumes. This value is unique among the volumes of a given server for that session.
`kFPVolBytesFreeBit` (bit 6)	Four unsigned bytes	Total bytes free on volumes less than 4 GB in size. If a volume is more than 4 GB, the Bytes Free parameters may not reflect the actual value. In any case, Extended Bytes Total always reflects the correct value. This value is maintained by the server and cannot be modified by an AFP client.
`kFPVolBytesTotalBit` (bit 7)	Four unsigned bytes	Total bytes on volumes less than 4 GB in size. If a volume is more than 4 GB, the Bytes Total parameter may not reflect the actual value. In any case, Extended Bytes Total always reflects the correct value. This value is maintained by the server and cannot be modified by an AFP client.
`kFPVolNameBit` (bit 8)	A string of up to 2 7 characters	The volume name identifies a server volume to an AFP client user, so it must be unique among all volumes managed by the server. All eight-bit ASCII characters, except null (0x00) and colon (0x3A), are permitted in a volume name. This name is not used directly to specify files and directories on the volume. Instead, the AFP client sends an AFP command to obtain a particular volume identifier, which it then uses when sending subsequent AFP commands. For more information, see "Designating a Path to a CNode" (page 33).
`kFPVolExtBytesFreeBit` (bit 9)	Eight unsigned bytes	Total bytes free on this volume. This value is maintained by the server and cannot be modified by an AFP client
`kFPVolExtBytesTotalBit` (bit 10)	Eight unsigned bytes	Total bytes on this volume. This value is maintained by the server and cannot be modified by an AFP client.

Constant and bit position	Parameter size	Description
kFPVolBlockSizeBit (bit 11)	Four bytes	The block allocation size.

The Attributes parameter for volumes provides additional information about the volume. Table 1-3 lists the bit definitions for the Attributes parameter for volumes.

Table 1-3 Bit definitions for the volume Attributes parameter

Constant and bit position	Description
kReadOnly (bit 0)	If set, the volume is available for reading only.
kHasVolumePassword (bit 1)	If set, the volume has a volume password. Volume passwords were supported in prior versions of AFP; now the volume attributes reflect this information. This bit is the same as the HasPassword bit returned for each volume by FPGetSrvrParms.
kSupportsFileIDs (bit 2)	If set, the volume supports file IDs. In general, if file IDs are supported on one volume, they are supported on all volumes, but this bit allows the server to be more selective, if necessary.
kSupportsCatSearch (bit 3)	If set, the volume supports the FPCatSearch and FPCatSearchExt commands. Support for FPCatSearch and FPCatSearchExt is optional. This bit allows the server to make this capability available on a per-volume basis.
kSupportsBlankAccessPrivs (bit 4)	If set, the volume has a Supports Blank Access Privileges bit that, when set for a directory, causes the directory to inherit its access privileges from its parent directory.
kSupportsUnixPrivs (bit 5)	If set, the volume supports UNIX privileges.
kSupportsUTF8Names (bit 6)	If set, the volume supports Unicode user names, group names, and pathnames.
kNoNetworkUserIDs (bit 7)	If set, always map UNIX user IDs, group IDs and permissions to traditional User IDs, Group IDs and permissions. If not set, after logging into the server, an AFP client running on a UNIX-based machine should call getuid() to get the user's local user ID and send an FPGetUserInfo command to get the user's user ID from the server. If the user IDs match, the AFP client should call getpwuid() to get the user's local user name, which is returned in the pw_name field, and send an FPMapID command to get the user's user name from the server. If the user names match, the AFP client assumes both

machines are operating from a common user directory, and displays UNIX permissions without mapping them. Showing UNIX user IDs, group IDs, and permissions is useful for home directory servers and other servers participating in a network user database. If the user IDs or user names do not match, or if the AFP client is not running on a UNIX-based machine, the AFP client should map UNIX user IDs, group IDs and permissions to traditional User IDs, Group IDs and permissions. This default behavior can be changed by settings on the server. The server can be forced to always set or to never set the `kNoNetworkUserIDs` bit.

Volume Types

An AFP volume is structured hierarchically. There are two types of hierarchical volumes: fixed and variable. A fixed Directory ID volume contains multiple directories, with each directory having its own permanent Directory ID that is assigned when the directory is created. The Directory ID is not used for any other directory during the lifetime of the volume, even if the directory to which it is assigned is later deleted.

A variable Directory ID volume also maintains the uniqueness of its Directory IDs but differs from a fixed Directory ID volume in that it does not associate a permanent Directory ID with each directory. For variable Directory ID volumes, the file server creates a unique Directory ID for a directory whenever the AFP client sends an `FPOpenDir` command. The file server then maintains this Directory ID until the client sends an `FPCloseDir` command or the AFP session terminates. A Directory ID obtained by sending an `FPOpenDir` command to a variable Directory ID volume must be used only for that session. If the Directory ID is stored and used to reference the directory in a later session, the results cannot be predicted: the command may fail, manipulate the wrong directory, or accidentally manipulate the correct directory.

Table 1-4 Volume types

Value	Description
1	Flat
2	Fixed Directory ID
3	Variable Directory ID (deprecated)

The volume types have the following support capabilities and constraints: Personal computers using MS-DOS can gain access to any type of server volume because the concept of Directory IDs is foreign to their file systems. However, Macintosh computers using the hierarchical file system (HFS) cannot directly use variable Directory ID volumes. Macintosh HFS volumes are fixed Directory ID volumes and hierarchical volumes on the file server can be handled by HFS only if they are fixed Directory ID volumes. Mac OS applications, such as the Finder, save Directory IDs and do no expect them to vary.

Note: AFP 3.*x* servers do not advertise support for variable Directory ID volumes, and AFP 3.*x* clients are not required to support variable Directory ID volumes.

Volume Catalog

The volume catalog is the structure that describes the branching tree arrangement of files and directories on fixed and variable Directory ID volumes. The catalog does not span multiple volumes; the AFP client sees a separate volume catalog for each server volume that is visible to AFP clients. Figure 1-2 (page 21) shows an example of a volume catalog and illustrates its elements.

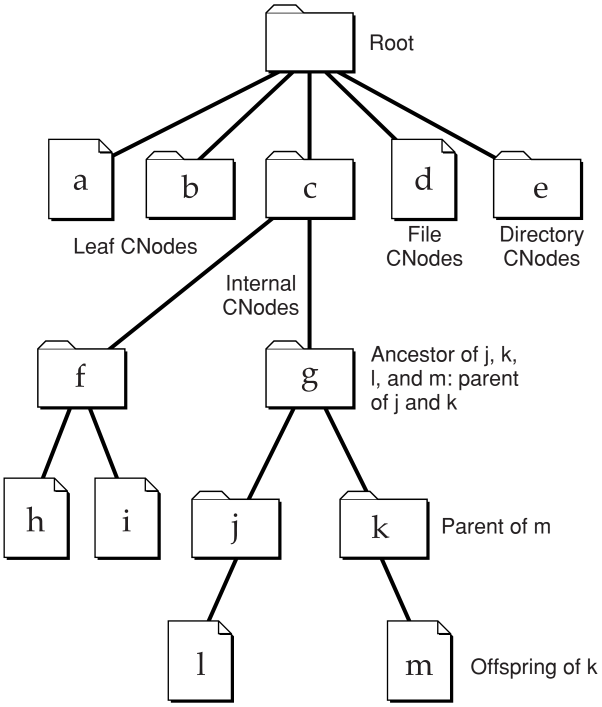
The volume catalog contains directories and files branching from a base directory known as the root. These directories and files are referred to as catalog nodes or CNodes (not to be confused with devices on a network, which are also known as nodes). Within the tree structure, CNodes can be positioned in two ways:

- at the end of a limb, in which case the CNode is called a leaf; a leaf CNode can be a file or an empty directory
- connected from above and below to other CNodes, which case the CNode is called an internal CNode. Internal CNodes are always directories

CNodes have a parent/offspring relationship. A give CNode is the offspring of the CNode above it in the catalog tree, and the higher CNode is considered its parent directory. Offspring are contained within the parent directory. The only CNode that does not have a parent directory is the root directory.

When an AFP command makes its way through the volume catalog, it can take only one shortest path from the root to a specific CNode. The CNodes along that path are said to be ancestors of the destination node, which in turn is called the descendent of each of its ancestors.

Figure 1-2 Volume catalog



Catalog Node Names

CNode names identify every directory and file in a volume catalog. Each directory and file has a long name and a short name, and may also have a Unicode name.

Long names and short names correspond in two of the native file systems that AFP supports: the Mac OS refers to files and directories by long names; MS-DOS computers use the short name format.

Unicode names conform to the Unicode Standard, which uses16-bits to encode more than 65,000 characters. To keep character coding simple and efficient, the Unicode Standard assigns each character a unique numeric value and name. To aid in conversion from Unicode to other script systems, Unicode Cnode names include a four-byte text encoding hint that specifies the script that was originally used to compose the Cnode name. The header file Text Encoding Conversion Manager (`TextCommon.h`) defines the constants that can be used as the text encoding hint:

```
enum {
    kTextEncodingMacRoman = 0,
    kTextEncodingMacJapanese = 1,
    kTextEncodingMacChineseTrad = 2,
    kTextEncodingMacKorean = 3,
    kTextEncodingMacArabic = 4,
    kTextEncodingMacHebrew = 5,
    kTextEncodingMacGreek = 6,
    kTextEncodingMacCyrillic = 7,
    kTextEncodingMacDevanagari = 9,
    kTextEncodingMacGurmukhi = 10,
    kTextEncodingMacGujarati = 11,
    kTextEncodingMacOriya = 12,
    kTextEncodingMacBengali = 13,
    kTextEncodingMacTamil = 14,
    kTextEncodingMacTelugu = 15,
    kTextEncodingMacKannada = 16,
    kTextEncodingMacMalayalam = 17,
    kTextEncodingMacSinhalese = 18,
    kTextEncodingMacBurmese = 19,
    kTextEncodingMacKhmer = 20,
    kTextEncodingMacThai = 21,
    kTextEncodingMacLaotian = 22,
    kTextEncodingMacGeorgian = 23,
    kTextEncodingMacArmenian = 24,
    kTextEncodingMacChineseSimp = 25,
    kTextEncodingMacTibetan = 26,
    kTextEncodingMacMongolian = 27,
    kTextEncodingMacEthiopic = 28,
    kTextEncodingMacCentralEurRoman = 29,
    kTextEncodingMacVietnamese = 30,
    kTextEncodingMacExtArabic = 31,
    kTextEncodingMacSymbol = 33,
    kTextEncodingMacDingbats = 34,
    kTextEncodingMacTurkish = 35,
    kTextEncodingMacCroatian = 36,
    kTextEncodingMacIcelandic = 37,
    kTextEncodingMacRomanian = 38,
    kTextEncodingMacCeltic = 39,
    kTextEncodingMacGaelic = 40,
    kTextEncodingMacKeyboardGlyphs = 41,
    kTextEncodingMacUnicode = 126,
    kTextEncodingMacFarsi = 140,
    kTextEncodingMacUkrainian = 152,
    kTextEncodingMacInuit = 236,
    kTextEncodingMacVT100 = 252,
    kTextEncodingMacHFS = 255,
    kTextEncodingUnicodeDefault = 256,
    kTextEncodingUnicodeV1_1 = 257,
    kTextEncodingISO10646_1993 = 257,
    kTextEncodingUnicodeV2_0 = 259,
    kTextEncodingUnicodeV2_1 = 259,
    kTextEncodingUnicodeV3_0 = 260,
    kTextEncodingISOLatin1 = 513,
    kTextEncodingISOLatin2 = 514,
    kTextEncodingISOLatin3 = 515,
    kTextEncodingISOLatin4 = 516,
    kTextEncodingISOLatinCyrillic = 517,
    kTextEncodingISOLatinArabic = 518,
    kTextEncodingISOLatinGreek = 519,
    kTextEncodingISOLatinHebrew = 520,
    kTextEncodingISOLatin5 = 521,
    kTextEncodingISOLatin6 = 522,
    kTextEncodingISOLatin7 = 525,
    kTextEncodingISOLatin8 = 526,
    kTextEncodingISOLatin9 = 527,
    kTextEncodingDOSLatinUS = 1024,
    kTextEncodingDOSGreek = 1029,
    kTextEncodingDOSBalticRim = 1030,
    kTextEncodingDOSLatin1 = 1040,
    kTextEncodingDOSGreek1 = 1041,
    kTextEncodingDOSLatin2 = 1042,
    kTextEncodingDOSCyrillic = 1043,
    kTextEncodingDOSTurkish = 1044,
    kTextEncodingDOSPortuguese = 1045,
    kTextEncodingDOSIcelandic = 1046,
    kTextEncodingDOSHebrew = 1047,
    kTextEncodingDOSCanadianFrench = 1048,
    kTextEncodingDOSArabic = 1049,
    kTextEncodingDOSNordic = 1050,
    kTextEncodingDOSRussian = 1051,
    kTextEncodingDOSGreek2 = 1052,
    kTextEncodingDOSThai = 1053,
    kTextEncodingDOSJapanese = 1056,
    kTextEncodingDOSChineseSimplif = 1057,
    kTextEncodingDOSKorean = 1058,
    kTextEncodingDOSChineseTrad = 1059,
    kTextEncodingWindowsLatin1 = 1280,
    kTextEncodingWindowsANSI = 1280,
    kTextEncodingWindowsLatin2 = 1281,
    kTextEncodingWindowsCyrillic = 1282,
    kTextEncodingWindowsGreek = 1283,
    kTextEncodingWindowsLatin5 = 1284,
    kTextEncodingWindowsHebrew = 1285,
    kTextEncodingWindowsArabic = 1286,
    kTextEncodingWindowsBalticRim = 1287,
    kTextEncodingWindowsVietnamese = 1288,
    kTextEncodingWindowsKoreanJohab = 1296,
    kTextEncodingUS_ASCII = 1536,
    kTextEncodingJIS_X0201_76 = 1568,
    kTextEncodingJIS_X0208_83 = 1569,
    kTextEncodingJIS_X0208_90 = 1570
};
```

To allow dissimilar computers to share resources, the file server provides CNode names in all three formats. When creating or renaming files and directories, the user provides a name consistent with the native file system. The server then uses an algorithm to generate the other name (long or short). This section describes the rules for forming CNode names and the algorithm used for creating and maintaining dual names.

The syntax for forming AFP long names is the same as the naming syntax used by the Macintosh HFS, with one exception: Null (0x00) is not a permissible character in AFP long names. Otherwise, the mapping of character code to character is the same for AFP as it is for Mac OS X. [See "Character Encoding" (page 68).] AFP long names are made up of at most 31 characters; valid characters are any printable ASCII code except colon (0x3A) and null (0x00). The volume name, and by inference the root's long name, cannot be longer than 27 bytes.

The syntax for forming AFP short names is the same as the naming syntax used by MS-DOS, which is more restrictive than the naming syntax used in the Mac OS: Names may be up to eight alphanumeric characters, optionally followed by a period (0x2E) and a one-to-three character alphanumeric character extension.

To ensure that a CNode can be uniquely specified by either name, AFP defines the following rules:

- no two offspring of a given directory can have the same short name or the same long name.

- a short name can match a long name if they both belong to the same file or directory.

Therefore, either name, long or short, uniquely identifies CNodes within the same directory.

AFP naming rules are such that any MS-DOS name can be used directly as a CNode short name, and any Mac OS X name can be used as a long name. The file server generates the other name for each CNode, deriving it from the first name specified and matching the second name as closely as possible. The long name format is a superset of the short name format. The name management algorithm mandates that whenever a CNode is created or renamed with a short name, the long name will always match. Deriving a short name from a long name is not so simple, and AFP does not stipulate an exact algorithm for this derivation. Therefore, different servers may perform this short-name creation differently.

When a CNode is created, the caller supplies the node's name and a name type that indicates whether the name is in short or long format. The server then checks the name to verify that the name conforms to the accepted format. The algorithm that follows describes how servers assign short and long names to a CNode (referred to as an object of this algorithm).

```
IF name type is short or name is in short format
THEN check for new name in list of short names
    IF name already exists
    THEN return ObjectExists result
    ELSE set object's short and long names to new name

ELSE { name type is long OR name is in long format }
    check for new name in list of long names
    IF name already exists
    THEN reutrn ObjectExists result
    ELSE set object's long name to new name
        derive short name from long name
```

This algorithm is used for renaming as well as for creating new names. When a user renames an object, its other name is changed using the above algorithm.

One limitation of this algorithm is that it does not prevent a user from specifying a long name that matches the short name generated by the file server for another file. A server-generated short name is normally not visible to an user that sees only long names. If a user inadvertently specifies a long name that matches a short name, the command fails and the server returns a kFPObjectErr.

For example, for a file created with the long name "MacFileLongName", a file server can generate a short name of "MacFile". When the user tries to create a new file with the long name "MacFile" in the same directory, the command fails because the above algorithm stipulates that the long name and the short name would both have to be set to "MacFile".

Note: The root directory of a volume catalog represents the volume, and the root's long name is the same as the volume name. The volume may also have a Unicode name. The volume has a short name, which is the short name of the root directory, but AFP does not allow its use. Neither the root nor the volume can be deleted or renamed through AFP.

If an AFP client creates a file having a Unicode name, the file server is required to generate a valid long name and a valid short name for the file. The algorithm for generating long and short names for a file having a Unicode file name is beyond the scope of this specification.

Directories and Files

Directories and files are stored in volumes and constitute the next level of the file system structure visible to AFP clients. As was shown in Figure 1-2 (page 21), directories branch to files and other directories. Each directory has an identifier through which it and its offspring can be addressed. Therefore, directories can be thought of as logically containing their offspring directories and files with the parameters described below.

Directory IDs

Each directory in the volume catalog is identified by a four-byte long integer known as its Directory ID. Because two directories on the same volume cannot have the same Directory ID, the Directory ID uniquely identifies a directory within a volume.

Within the volume catalog, as mentioned earlier, directories have ancestor, parent, and offspring relationships with each other. The Directory ID of a CNode's parent is called the CNode's Parent ID.

A CNode can have only one parent, so a give CNode has a unique Parent ID. However, a CNode can have several ancestor directory identifiers, one for each ancestor. The parent directory is considered an ancestor.

Directory IDs from 1 to 16 are reserved. The Directory ID of the root is always 2. The root's Parent ID is always 1. (The root does not really have a parent; this value is returned only if an AFP command asks for the root's Parent ID.) Zero (0) is not a valid Directory ID.

Directory Parameters

For each directory, the server must maintain the parameters listed in Table 1-5. The parameters are obtained by calling `FPGetFileDirParms` and specifying in the `DirBitMap` parameter the directory parameters that are to be obtained. Some directory parameters can be set by calling `FPSetDirParms` or `FPSetFileDirParms`.

Table 1-5 Definitions for the Directory bitmap

Constant and bit position	Parameter size	Description
`kFPAttributeBit` (bit 0)	Two bytes	Additional information about the directory. For details, see Table 1-6 (page 27).
`kFPParentDirIDBit` (bit 1)	Four bytes	The unique identifier of the directory's parent directory.
`kFPCreateDateBit` (bit 2)	Four bytes	Date the directory was created. For more details, see the section "Date-Time Values" (page 31).

Constant and bit position	Parameter size	Description
kFPModDateBit (bit 3)	Four bytes	Date the directory was most recently modified. For more details, see the section "Date-Time Values" (page 31).
kFPBackupDateBit (bit 4)	Four bytes	Date the directory was most recently backed up. For more details, see the section "Date-Time Values" (page 31).
kFPFinderInfoBit (bit 5)	32 bytes	Accompanies directories that are used by computers with HFS. This parameter is maintained by the AFP client and is not examined by AFP.
kFPLongNameBit (bit 6)	String of up to 32 characters	The directory's long name. For details, see "Catalog Node Names" (page 21).
kFPShortNameBit (bit 7)	String of up to 12 characters	The directory's short name. For details, see "Catalog Node Names" (page 21).
kFPNodeIDBit (bit 8)	Four bytes	The directory's unique identifier.
kFPOffspringCountBit (bit 9)	Four bytes	Number of files and directories contained by the directory.
kFPOwnerIDBit (bit 10)	Four bytes	A User ID that uniquely identifies the owner of the directory. Starting with AFP 2.0, a directory can be owned by a group. For more information about the Owner ID parameter, see the section "Directory Access Controls" (page 61).
kFPGroupIDBit (bit 11)	Four bytes	A number that uniquely identifies the group affiliation of the directory. Starting with AFP 2.0, the Group ID can be a User ID. For more information about the Group ID parameter, see the section "Directory Access Controls" (page 61).
kFPAccessRightsBit (bit 12)	Four bytes	A bitmap that describes the access rights for the directory's owner, group affiliation, and Everyone. This bitmap also includes the UARights Summary byte.
kFPUTF8NameBit (bit 13)	AFP Name	The directory's name in Unicode format. For information about Unicode format, see "Catalog Node Names" (page 21).
kFPUnixPrivsBit (bit 15)	16 bytes	If the directory has UNIX privileges, they are stored in an FPUnixPrivs structure.

The Attributes parameter for directories provides additional information about the directory. Table 1-6 lists the bit definitions for the Attributes parameter for directories.

Table 1-6 Bit definitions for the directory Attributes parameter

Bit	Description
Invisible (bit 0)	The directory should not be made visible to the user.
IsExpFolder (bit 1)	The directory is a share point. This directory, and all directories within it, indicate to the user that access privileges are valid (for example, by displaying tabbed folders or drop-box folder icons or by enabling the Sharing menu item). None of the directories outside the shared (exported) area show access privileges on local computers, although they may still have valid access privilege information that only an administrator can see or modify. This bit is a read only bit. It cannot be set by `FPSetFileDirParms`.
System (bit 2)	The directory is a system directory; the definition of "system directory" is left to the local computer.
Mounted (bit 3)	The directory 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 directory is a share point, and that a remote user currently has it mounted. This bit is a read only bit. It cannot be set by `FPSetFileDirParms`.
InExpFolder (bit 4)	The directory is in a shared area. This directory, and all directories within this directory, indicate to the user that access privileges are valid. This directory cannot be shared because a share point cannot exist within another share point. This bit is a read only bit. It cannot be set by `FPSetFileDirParms`.
BackupNeeded (bit 6)	The directory needs to be backed up. This bit is set whenever the directory's modification date-time is modified.
RenameInhibit (bit 7)	The directory cannot be renamed.
DeleteInhibit (bit 8)	The directory cannot be deleted.
Set/Clear (bit 15)	When used in conjunction with the `FPSetFileDirParms` command, indicates whether the specified attributes are to be set (1) or cleared (0). It is not possible to set some attributes and clear other attributes in the same call.

No specific bit exists to inhibit moving a directory, but directory movement is constrained by the RenameInhibit bit when a directory is moved or moved and renamed.

Access Rights (a four-byte quantity) contains the read, write, and search access privileges corresponding to the directory's owner, group, and Everyone. The upper byte of the Access Rights parameter is the User Access Rights Summary byte, which indicates the privileges the current user of the AFP client has to this directory. Table 1-7 lists the bit definitions for the Access Rights parameter for directories.

Table 1-7 Bit definitions for the directory Access Rights parameter

Bit	Description
(bit 0)	Set if the directory's owner has search access to this directory.
(bit 1)	Set if the directory's owner has read access to this directory.
(bit 2)	Set if the directory's owner has write access to this directory.
(bit 8)	Set if the directory's group has search access to this directory.
(bit 9)	Set if the directory's group has read access to this directory.
(bit 10)	Set if the directory's group has write access to this directory.
(bit 16)	Set if Everyone has search access to this directory.
(bit 17)	Set if Everyone has read access to this directory.
(bit 18)	Set if Everyone has write access to this directory.
(bit 24)	Set if the user has search access to this directory.
(bit 25)	Set if the user has read access to this directory.
(bit 26)	Set if the user has write access to this directory.
Blank Access Privileges (bit 28)	This directory has blank access privileges and has the same access privileges as the directory enclosing it.
(bit 31)	Set if the user is the owner of the directory. It is also set if the directory is not owned by a registered user.

An `FPUnixPrivs` structure is used to return UNIX privileges if a file or directory resides on a volume that supports UNIX privileges. The `FPUnixPrivs` structure is defined as

```
struct FPUnixPrivs {
    unsigned long uid;
    unsigned long gid;
    unsigned long permissions;
    unsigned long ua_permissions;
};
```

where

- `uid` is the user ID of the file or directory's owner
- `gid` is the group ID of the file or directory's owner
- `permissions` is the setting of the file or directory's permission bits
- `ua_permissions` is the user's access rights to the file or directory; bit 31 is set if the user is the owner of the file or directory

File Parameters

For each file, the server must maintain the parameters listed in Table 1-8. The parameters are obtained by calling `FPGetFileDirParms` and specifying in the `FileBitmap` parameter the file parameters that are to be obtained, by calling `FPResolveID` and specifying the file's File ID, or by calling `FPGetForkParms`. Some file parameters can be set by sending `FPSetFileParms`, `FPSetFileDirParms`, and `FPSetForkParms` commands.

Table 1-8 Definitions for the File bitmap

Constant and bit position	Parameter size	Description
`kFPAttributeBit` (bit 0)	Two bytes	Additional information about the file. For details, see Table 1-9 (page 30).
`KFPParentDirIDBit` (bit 1)	Four bytes	The unique identifier for the file's parent directory.
`kFPCreateDateBit` (bit 2)	Four bytes	Date the file was created. For more details, see the section "Date-Time Values" (page 31).
`kFPModDateBit` (bit 3)	Four bytes	Date the file was most recently modified. For more details, see the section "Date-Time Values" (page 31).
`kFPBackupDateBit` (bit 4)	Four bytes	Date the file was most recently backed up. For more details, see the section "Date-Time Values" (page 31).
`kFPFinderInfoBit` (bit 5)	32 bytes	Accompanies files that are used by computers with HFS. This parameter is maintained by the AFP client and is not examined by AFP.
`kFPLongNameBit` (bit 6)	String of up to 32 characters	The file's long name. For details, see "Catalog Node Names" (page 21).
`kFPShortNameBit` (bit 7)	String of up to 12 characters	The file's short name. For details, see "Catalog Node Names" (page 21).
`kFPNodeIDBit` (bit 8)	Four bytes	The file's unique number obtained by the file server from the File Manager's PBGetCatInfo call.
`kFPDataForkLenBit` (bit 9)	Four-byte unsigned integer	The current length of the file's data fork. If the data fork's length is greater than 4 GB, specifying this bit returns the actual length of the data fork. If the data fork's length is greater than 4 GB, specifying this bit returns 4 GB.

Constant and bit position	Parameter size	Description
`kFPRsrcForkLenBit` (bit 10)	Four-byte unsigned integer	The current length of the file's resource fork. If the resource fork's length is greater than 4 GB, specifying this bit returns the actual length of the resource fork. If the resource fork's length is greater than 4 GB, specifying this bit returns 4 GB.
`kFPExtDataForkLenBit` (bit 11)	Eight-byte unsigned integer	The current length of the file's data fork.
`kFPLaunchLimitBit` (bit 12)		
`kFPUTF8NameBit` (bit 13)	AFPName	The file's name in Unicode format.
`kFPExtRsrcForkLenBit` (bit 14)	Eight-byte unsigned integer	The current length of the file's resource fork.
`kFPUnixPrivsBit` (bit 15)	16 bytes	If the file has UNIX privileges, they are returned in an `FPUnixPrivs` structure.

The file number is a unique number associated with each file on the volume. This number is purely informative; AFP does not allow the specification of a file by its file number.

The Attributes parameter for files provides additional information about the file. Table 1-9 lists the bit definitions for the Attributes parameter for files.

Table 1-9 Bit definitions for file Attributes parameter

Constant and bit position	Description
`kFPInvisibleBit` (bit 0)	File should not be made visible to the user.
`kFPMultiUserBit` (bit 1)	File is an application that has been written for simultaneous use by more than one user.
`kFPSystemBit` (bit 2)	File is a system file.
`kFPDAlreadyOpenBit` (bit 3)	File's data fork is currently open by a user.
`kFPRAlreadyOpenBit` (bit 4)	File's resource fork is currently open by a user.
`kFPWriteInhibitBit` (bit 5)	User cannot write to the file's forks.

Constant and bit position	Description
kFPBackUpNeededBit (bit 6)	File needs to be backed up.
kFPRenameInhibitBit (bit 7)	File cannot be renamed.
kFPDeleteInhibitBit (bit 8)	File cannot be deleted.
kFPCopyProtectBit (bit 10)	File should not be copied.
kFPSetClearBit (bit 15)	When used in conjunction with the FPSetFileDirParms command, indicates whether the specified attributes are to be set (1) or cleared (0). It is not possible to set some attributes and clear other attributes in the same call.

No specific bit exists to inhibit moving a file, but file movement is constrained by the RenameInhibit bit only when a file is moved and renamed, not when it is simply moved.

The Finder will not copy a file whose CopyProtect bit is set. An attempt to copy the file using the FPCopyFile request will in an error. This bit may be read, but not set, using AFP. It is to be set by some administrative program, whose specification is beyond the scope of this document.

The BackupNeeded bit is set whenever the file's modification date-time changes.

The data fork length and resource fork length are equal to the number of bytes in the corresponding fork.

The creation, backup, and modification date-time parameters are described next.

Date-Time Values

All date-time quantities used by AFP specify values of the server's clock. These values correspond to the number of seconds measured from 12:00 am on January 1, 2000 in Greenwich Mean Time (GMT). AFP represents date-time values with four-byte signed integers. One of the AFP commands allows the AFP client to obtain the current value of the server's clock. At login time, the AFP client should read this value (*s*) and the value of the AFP client's clock (*w*) and computer the offset between these values (*s* - *w*). All subsequent date-time values read from the server should be adjusted by adding this offset to the date-time. This adjustment will correct for differences between the two clocks and will ensure that all computers see a consistent time base.

The creation date-time of a directory or a file is set to the server's system clock when the file or directory is created. The backup date-time is set by backup programs. When a file or directory is created, the server sets the backup date-time to 0x80000000, which is the earliest representable time.

The server changes the modification date-time of a directory each time the directory's contents are modified. Therefore, any of the following actions will cause the server to assign a new modification date to the directory: renaming the directory; creating or deleting a CNode in the directory; moving the directory; changing its access privileges, Finder Info, or changing the Invisible attributes of one of its offspring.

An AFP client with the appropriate privileges can set the creation and modification date-time parameters to any value.

File Forks

A file consists of two forks: a data fork and a resource fork. The bytes in a file fork are sequentially numbered starting with 0. The data fork is an unstructured finite sequence of bytes. The resource fork is used to hold Mac OS resources, such as icons and drivers, and a data structure for mapping them within the fork. AFP is designed to consider both forks as finite-length byte sequences; however, AFP contains no rules relating to the structure of the resource fork. For more information about resource forks, refer to *Inside Mac OS X*.

Either or both forks of a given file can be empty. Non-Mac OS AFP clients that need only one file fork must use the data fork. Files created by a computer with an MS-DOS operating system will have an empty resource fork because a resource fork is unintelligible to that operating system. Consequently, an MS-DOS computer that has gained access to a server file created by a Macintosh may not be aware of the existence of the file's resource fork.

Although AFP allows the creation of MS-DOS applications that can understand and manipulate resource forks, such applications would have to preserve the internal structure of the forks. Mac OS computers expect a specific format in the resource fork of any file, so AFP clients of computers that cannot manage the internal structure of the resource fork should never alter the contents of a resource fork.

To read from or write to the contents of a file's data or resource fork, the AFP client first sends a command to open the particular fork of the file, creating an access path to that file fork. The access path is not be confused with the paths and pathnames described in the next section.

Once the AFP client creates this access path, all subsequent read and write commands refer to it for the duration of the session. For each access path, the server maintains the parameters listed in Table 1-10.

Table 1-10 Access path parameters

Parameter	Description
`OForkRefNum`	Two bytes (0 is invalid)
`AccessMode`	Two-byte bitmap
`Flag`	Bit 7 of a one-byte value

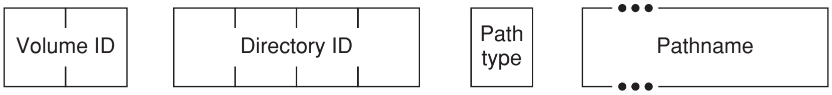
The `OForkRefNum` parameter uniquely identifies the access path among all access paths within a given session. The `AccessMode` parameter indicates to the server whether this access path allows reading or writing. It is maintained by the server and is inaccessible to clients of AFP. The `Flag` parameter indicates to the server that the access path belongs to the data or the resource fork.

In addition to the above parameters, the server must provide a way to gain access to the parameters of the file to which an open fork belongs. For details, see the `FPGetForkParms` command in the Reference section.

Designating a Path to a CNode

In order to perform any action on a CNode, the AFP client must designate a path to the CNode. AFP provides rules for specifying a path to any CNode in the volume catalog. A CNode (file or directory) can be unambiguously specified to the server by the identifiers shown in Figure 1-3.

Figure 1-3 CNode specification



The Volume ID specifies the volume on which the destination CNode resides. The Directory ID can belong to the destination CNode (if the CNode is a directory) or to any one of its ancestor directories, up to and including the root directory and the root's parent directory.

An AFP pathname is formatted as a Pascal string (one length byte followed by the number of characters specified by the length byte) or a Unicode string (a four-byte text encoding hint followed by two length bytes followed by the number of characters specified by the length bytes). An AFP pathname is made up of CNode names, concatenated with intervening null-byte separators. Each element of a pathname must be the name of a directory, except for the last one, which can be the name of a directory or a file.

The elements of a pathname can be long or short names. However, a given pathname cannot contain a mixture of long and short names. A path type byte, which indicates whether the elements of the pathname are all short or all long names, is associated with each pathname. A pathname consisting of short names and a path type of 1. A pathname consisting of long names has a path type of 2.

An AFP pathname that consists of long or short names can be up to 255 characters long. The length of an AFP pathname that consists of Unicode names is virtually unlimited. A single null byte as the length byte indicates that no pathname is supplied. Because the length byte is included at the beginning of the string, each pathname element (CNode name) does not include a length indicator.

The syntax of an AFP pathname follows this paragraph. The asterisk (*) represents a sequence of zero or more of the preceding elements of the pathname; the plus (+) represents a sequence of one or more of the preceding elements; `<Sep>` represents the separators in the pathname; the vertical bar (|) is an OR operator; and the term on the left side of the ::= symbol is defined as the term(s) on the right side.

```
<Sep> ::= <null-byte>+
<Pathname> ::= empty-string |
    <Sep>*<CNode name>(<Sep><Pathname>*
```

The syntax represents a concatenation of CNode names separated by one or more null bytes. Pathnames can also start or end with a string of null bytes.

A pathname can be used to traverse the volume catalog in any direction. The pathname syntax allows paths either to descend from a particular CNode through its offspring or to ascend from a CNode to its ancestors. In either case, the directory that is the starting point of this path is defined separately from the pathname by its Directory ID. The first element of the pathname is an offspring of the starting point of the directory. The pathname must be parsed from left to right to obtain each element that is used as the next node on the path.

To descend through a volume, a valid pathname must proceed in order from parent to offspring. A single null-byte separator preceding this first element is ignored.

To ascend through a volume, a valid pathname must proceed from a particular CNode to its ancestor. To ascend one level in the catalog tree, two consecutive null bytes should follow the offspring CNode name. To ascend two levels in the catalog tree, three consecutive null bytes are used as the separator, and so on.

A particular volume may descend and ascend through the volume catalog. Because of this, many valid pathnames may refer to the same CNode.

A complete path specification can take a number of forms. The table that follows summarizes the different kinds of path specifications that can be used to traverse the volume catalog illustrated in Figure 1-4 (page 34). A zero in square brackets `[0]` represents a null-byte separator.

The descriptors and examples that follow refer to this table and the corresponding volume catalog illustrated in Figure 1-4. To simplify these examples, the CNodes in this catalog are named *a* through *j*, except the root, which is named *x*. The path type is ignored in this example. The letter *v* represents the volume's two-byte Volume ID. Lines connect the CNodes; the unconnected lines indicate that other CNodes in this volume are not shown.

Figure 1-4 Example of a volume catalog

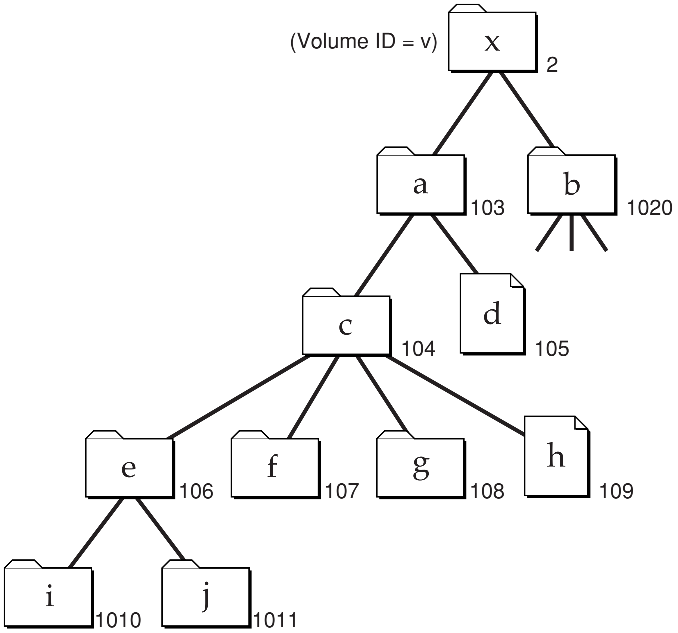


Table 1-11 provides the Volume ID, Directory ID, and pathname for some sample path specifications in Figure 1-4 (page 34).

Table 1-11 Sample path specifications

Example	Volume ID	Directory IDs	Pathname
First	v	2	a[0]c[0]e[0]j[o]
Second	v	104	e[0]j
Third	v	106	[0]j
Fourth	v	106	j
Fifth	v	106	[0]
Sixth	v	104	e[0][0]g[0][0]h
Seventh	v	104	e[0][0][0]
Eighth	v	1	x[0]a[0]c[0]h

The first example of a path specification in Table 1-11 contains the Volume ID, the root directory's Directory ID, which is always 2, and a pathname. In this case, the pathname must contain the names of all of the destination file's ancestors except the root, and it must end with the name of the file itself. The single trailing null byte is ignored.

The second example contains the Volume ID, the Directory ID of an ancestor, and a pathname.

The third example is essentially the same as the second example. The single leading null byte is ignored.

In the fourth example, the Directory ID is the Parent ID of the destination file. In this case, the pathname need contain only the name of the destination file itself.

The fifth example illustrates another way to uniquely specify a descending path to a directory. It includes the CNode's Volume ID, its Directory ID, and a null pathname. This path specification is used to specify the directory *e*.

The sixth example illustrates a descending path. The first CNode in the pathname is the offspring of the starting point Directory ID. Then the pathname ascends though *e*'s parent (c) down to directory *g*, backup to *g*'s parent (*c*), and down again to *h*.

The seventh shows an ascending pathname that starts at directory *c* (whose Directory ID is 104), moves down to *e*, and then ascends to *e*'s parent's parent (*a*).

The eighth example is a special case in which the starting point of the path is Directory ID 1, the parent of the root. The first name of the pathname must be the volume name or root directory name corresponding to Volume ID *v*; beyond that, pathname traversal is performed as in the other examples.

AFP Login

To make use of any resource managed by a file server, the AFP client must first log in to the server. This section provides an overview of the AFP login process.

After a user selects an AFP server to log in to, the AFP client sends the FPGetSrvrInfo command to request information about that server. The server returns information that includes

- the AFP versions the server supports
- the user authentication methods (UAMs) the server supports
- the server's machine type
- the server's name
- the server's network address
- the server's signature
- whether the server supports optional functionality, such as reconnect, Open Directory, FPCopyFile, FPChangePassword, saving passwords, and server notifications

During the AFP login process, the AFP client tells the server which AFP version the client will use to establish the connection and which UAM it will use to authenticate the user.

Each AFP version is uniquely described by a string of up to 16 characters called the AFP version string. The AFP version strings for the AFP versions supported by AFP 3.*x* are listed in Table 1-12 (page 36).

Table 1-12 AFP version strings

AFP version	AFP version string
AFP 2.1	AFPVersion 2.1
AFP 2.2	AFP2.2
AFP 2.3	AFP2.3
AFP 3.0	AFPX03
AFP 3.1	AFP3.1

The UAMs supported by AFP 3.*x* and their corresponding strings are listed in Table 1-13 (page 36).

Table 1-13 AFP UAM strings

String	UAM name
`No User Authent`	No User Authentication UAM. For details, see "No User Authentication" (page 41).
`Cleartxt Passwrd`	Cleartext Password. For details, see "Cleartext Password" (page 41).

String	UAM name
Randum Exchange	Random Number Exchange. For details, see "Random Number Exchange" (page 42).
2-Way Randnum	Two-Way Random Number Exchange. For details, see "Two-Way Random Number Exchange" (page 44).
DHCAST128	Diffie-Hellman Exchange. Allows the client to send a password of up to 64 bytes to the server through a strongly encrypted "tunnel." This type of encryption is useful for servers that require the use of cleartext password. For details, see "Diffie-Hellman Exchange" (page 46).
DHX2	Diffie-Hellman Exchange 2. Allows the client to send a password of up to 256 bytes to the server through a strongly encrypted "tunnel." This type of encryption is useful for servers that require the use of cleartext password. For details, see "Diffie-Hellman Exchange 2" (page 50).
Client Krb v2	Kerberos. Allows the client to use Kerberos v4 and Kerberos v5 tickets to authenticate a user.
Recon1	The Reconnect UAM. Allows the client to use the FPLoginExt command to reconnect using a reconnect token (also known as a *credential*) containing all of the information required to authenticate.

The prospective AFP client initiates the login process by sending an FPLogin or an FPLoginExt command to the server. Both commands include the AFP version string and the UAM string that the client has selected. Depending on the selected UAM method, the FPLogin or FPLoginExt command may include user login information (such as a user name or password), or a subsequent FPLoginCont command may include such information. The sending of additional FPLoginCont commands may be required to complete user authentication, as described in the next section, "File Server Security" (page 40).

If the UAM succeeds, an AFP session between the AFP client and the server begins.

As mentioned earlier, in addition to the AFP and UAM versions that the server supports, the FPGetSrvrInfo command returns a Flags parameter whose bits provide additional information about the server that is useful to an AFP client. The bits of the Flags parameter are listed in Table 1-14 (page 37).

Table 1-14 Bit definitions for the Flags parameter returned by FPGetSrvrInfo

Constant and bit position	Description
kSupportsCopyFile (bit 0)	Set if the server supports the FPCopyFile command.
kSupportsChgPwd (bit 1)	Set if the server supports the FPChangePassword command.

Constant and bit position	Description
kDontAllowSavePwd (bit 2)	Set if 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.
kSupportsSrvrMsg (bit 3)	Set if the server supports the FPGetSrvrMsg command.
kSrvrSig (bit 4)	Server supports server signatures, which is a 16-byte number that uniquely identifies the server. An AFP client should use the server signature to ensure that it does not log in to the same server multiple times. Preventing multiple logins is important when the server is configured for multihoming.
kSupportsTCP (bit 5)	Set if the server supports TCP/IP.
kSupportsSrvrNotify (bit 6)	Set if the server supports server notifications.
kSupportsReconnect (bit 7)	Set if the server supports the FPGetSessionToken and FPDisconnectOldSession commands.
kSupportsDirServices (bit 8)	Set if the server supports Open Directory.
kSupportsUTF8Name (bit 9)	Set if the server provides an offset to the server's name in UTF-8 format in the reply block.

Reconnecting Sessions

If an AFP session is disconnected due, for example, a network outage, but the AFP client still has the required information, the AFP client can reconnect the session.

Clients that use the Reconnect UAM, described in "Reconnect" (page 56), follow these steps to reconnect:

1. Log in successfully by calling FPLoginExt using a UAM that provides a session key.
2. Call FPGetSessionToken to get a token, specifying the Type parameters as kRecon1Login (5).
3. Periodically call FPGetSessionToken with the Type parameter set to kReconn1Refresh (7) to refresh the token before it expires.
4. If a disconnect occurs, call FPLoginExt to log in again, specifying the Reconnect UAM as the UAM, and passing the current token obtained by calling FPGetSessionToken in step 2 or 3. The reconnect token contains all of the user name and password information required for the server to authenticate the client, so logging in again does not require the client to repeat the authentication steps that took place in step 1.

5. If the login in step 4 completes successfully, call `FPDisconnectOldSession` and pass the token obtained in step 2. If the server can find the previous session identified by the token, it will transfer all the previous session's open files and locked resources to the new session and return a result code of `kFPNoErr`.

Clients that don't use the Reconnect UAM follow these steps to reconnect:

1. Log in successfully by calling `FPLogin` or `FPLoginExt`.
2. Call `FPGetSessionToken` to get a token.
3. If a disconnect occurs, log in again using the same UAM, user name and password that were used in step 1.
4. Call `FPDisconnectOldSession` and pass the token obtained in step 2. If the server can find the previous session identified by the token, it will transfer all the previous session's open files and locked resources to the new session and return a result code of `kFPNoErr`.

In either case, if the reconnect fails for any reason, the client must start over by logging in again as in step 1. The client should call `FPDisconnectOldSession` and send the current token to tell the server that it can free resources that were locked up by the earlier session.

Note: The `FPDisconnectOldSession` command will fail if the server cannot find the previous session or if the AFP client does not use same information to log in again. For security reasons, the server also fails all reconnects if the user originally logged in as the Guest user.

If the server returns a result code other than `kFPNoErr`, the AFP client can attempt to reopen its files. If the files were previously opened without Deny Modes and the AFP client did not apply byte range locks, the client should be able to reopen those files. In this case, reconnect is also deemed successful. If the reconnect is not successful, an AFP client can take the steps described in the next section, "Recovering From a System Crash".

Recovering From a System Crash

If an AFP session is disconnected and the client reconnect information is lost due to a local system crash, the AFP client will not be able to reconnect the session. If the server allows reconnect, any files that were left open on the remote server when the local system crashed will be saved but will not be available for opening until the reconnect timeout expires. This also applies to the case where a sleeping AFP client fails to wake up or crashes and the server is saving the information until the sleep timeout expires.

To tell the server to close files left open by an old session and disconnect that session, an AFP client that supports AFP 3.1 can create and save a unique client-defined identifier and use the `FPGetSessionToken` command to send it the server. The client must do this before the local system crashes, for example, as part of its login sequence. When it receives the identifier, the server associates the identifier with the current session.

Later, if the local system crashes and is restarted, the AFP client can log in and send the `FPGetSessionToken` command again, this time telling the server to look for a session having the specified identifier. If the server finds such a session, it closes the files that are associated with it, frees any other associated resources, and disconnects the old session.

Note: For security purposes, before disconnecting the old session, the server verifies that the same login information was used to log in from the same system. For security purposes, the server also fails `FPGetSessionToken` if the user originally logged in as the Guest user.

Disconnect Timers

In previous versions of AFP, there was only one timer for determining whether a disconnect had occurred. With 10.2.x, there are two timers for determining disconnections:

- Active timer, which is set to 60 seconds by default
- Idle timer, which is set to 120 seconds by default

If the client has an outstanding request to the server and has not received any data (including tickles) from the server, the client waits until the idle timer expires before assuming that a disconnect has occurred.

If the client has no outstanding requests to the server, the client waits until the idle timer expires before assuming that a disconnect has occurred.

Special considerations arise when the system is awakening from sleep:

- If the system is on a LAN, the active timer is set to (activeTimer / 4).
- If the system is on a WAN, the active timer is set to (activeTimer / 2).

If the client is connected to an AFP 2.x server, the Active time and the Idle timer are both set to 120 seconds.

Note: The `check_afp.app` plist contains an option that can be set to disallow idle sleep if an AFP volume is mounted.

In all situations, after a disconnect, if the server supports reconnect, reconnect is started.

File Server Security

Information stored in a shared resource needs protection from unauthorized users. The role of file server security is to provide varying amounts and kinds of protection, depending on what users feel is necessary.

AFP provides security in three ways:

- user authentication when the user logs in to the server
- an optional volume-level password when the user first attempts to gain access to a volume
- directory access controls

User Authentication Methods

AFP provides the capability for servers and AFP clients to use a variety of methods to authenticate users when they log in. Five user authentication methods are defined: no user authentication, cleartext password, random number exchange, two-way random number exchange, and Diffie-Hellman Exchange. Some UAMs are also used to change a password after the user logs in.

The AFP client indicates its choice of UAM by giving the server a UAM string. These strings are intended to be case-insensitive and diacritical-sensitive.

Some UAMs require additional user authentication information to be passed to the server in the `FPLogin` or `FPLoginExt` command. The following sections describe the UAMs and the kinds of information they require.

No User Authentication

The No User Authentication UAM requires no authentication information. When the `FPLogin` and `FPLoginExt` commands use the No User Authentication UAM, there is no *UserAuthInfo* parameter. The corresponding case-insensitive UAM protocol name for the No User Authentication UAM is `No User Authent`. The No User Authentication UAM is used when a user logs on as the Guest user.

In order to implement the directory access controls described later in this section, the server must assign a User ID and a Group ID to the user for the session.

User ID numbers and Group ID numbers are assigned from the same pool of numbers. In addition, starting with AFP 2.1, AFP servers must assign zero to users who log in as the Guest user and 1 to the Administrator/Owner.

In this UAM, the server assigns to the user "Everyone" access privileges for every directory in every server volume. "Everyone" access privileges are described in the section "Directory Access Controls" (page 61).

Cleartext Password

The Cleartext Password UAM transmits the user name and password to the server as cleartext (that is, not encoded). The protocol name for the Cleartext Password UAM is `Cleartxt Passwrd`.

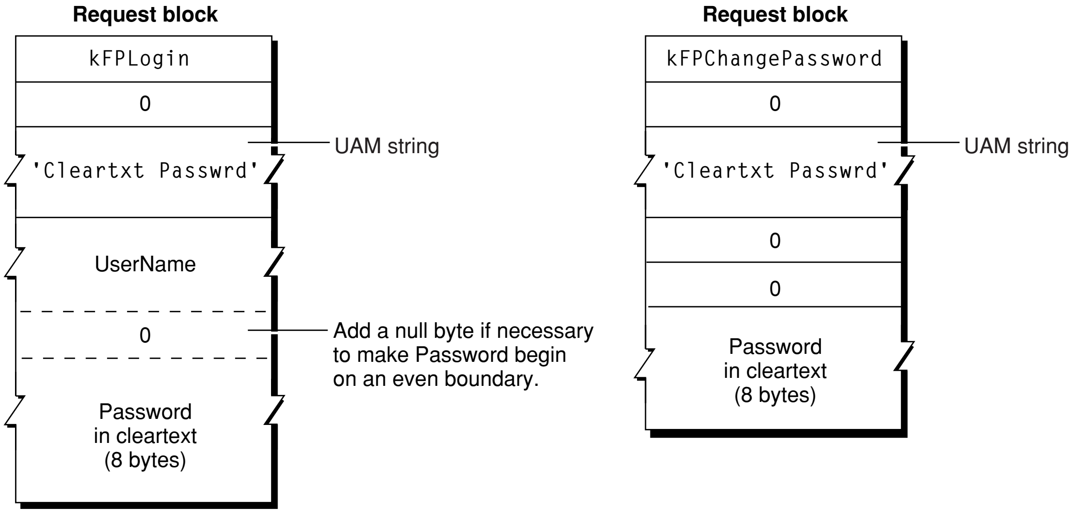
For the `FPLogin` command, the `UserAuthInfo` parameter consists of a user name (which is a string of up to 255 Macintosh Roman characters) followed by the user's password (up to 8 bytes). For the `FPLoginExt` command, the `UserAuthInfo` parameter consists of a user name (which is a string of up to 255 Unicode characters) followed by the user's password (up to 8 bytes). To ensure that the user's password is aligned on an even byte boundary in the packet, the AFP client may have to insert a null byte (0x00) between the user name and the password. If the user provides a password that is shorter than 8 bytes, it must be padded at the end with null bytes to make the password eight bytes long. The permissible set of characters in passwords consists of all 8-bit ASCII characters.

User name comparison is case-insensitive, but password comparison is case-sensitive for this UAM. If there is a user of the specified name whose password matches the password supplied by the caller of FPLogin or FPLoginExt, the user has been authenticated and the login succeeds. If the passwords do not match, a kFPUserNotAuth result code is returned.

The Cleartext Password UAM should be used by AFP clients only if the intervening network is secure against eavesdropping. Otherwise, the password information can be read from FPLogin or FPLoginExt command packets by anyone listening on the network.

Figure 1-5 (page 42) shows the request block for calling FPChangePassword when using the Cleartext Password UAM.

Figure 1-5 Request block when using the Cleartext Password UAM


Request block
kFPLogin
0
'Cleartxt Passwrd'
UserName
0
Password in cleartext (8 bytes)
UAM string
Add a null byte if necessary to make Password begin on an even boundary.
Request block
kFPChangePassword
0
'Cleartxt Passwrd'
0
0
Password in cleartext (8 bytes)
UAM string

Random Number Exchange

In environments in which the network is not secure against eavesdropping, Random Number Exchange is a more secure user authentication method. The protocol name for this UAM is Randum Exchange. With Random Number Exchange, the user's password is never sent over the network and cannot be picked up by eavesdropping. Deriving the password from information sent over the network is as difficult as breaking a DES-encrypted password.

With the Random Number Exchange UAM, only the user name is sent in the UserAuthInfo parameter of the FPLogin or FPLoginExt command. If the user name is valid, the server generates an eight-byte random number and sends it back to the AFP client, along with an ID number and a kFPAuthContinue result code. The kFPAuthContinue result code indicates that all is well at this point, but the user has not yet been authenticated. The AFP client then encrypts the random number with the user's password and sends the result to the server in the UserAuthInfo parameter of the FPLoginCont command along with the ID number returned earlier by the server in the reply block from the FPLogin or FPLoginExt command. The server uses the ID number to associate an earlier FPLogin or 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 kFPUserNotAuth result code.

The following steps explain the Random Number Exchange UAM in greater detail:

1. The AFP client sends the FPLogin or FPLoginExt command block with the UAM string and the UserAuthInfo parameter containing the user name string. For FPLogin, the user name can be up to 255 Macintosh Roman characters long; for FPLoginExt, the user name can be up to 255 Unicode characters long.

2. Upon receiving this command block, the server examines its user database to determine whether the user name is valid. User name comparison is case-insensitive and diacritical-sensitive.

3. If the server does not find the user name in the user database, it sends an error code to the AFP client indicating that the user name is not valid and denies the login request. If the server finds the name in the user database, it generates an eight-byte random number and sends it to the AFP client, along with an ID number and an kFPAuthContinue result code. The kFPAuthContinue result code indicates that all is well at this point, but the user is not yet authenticated.

4. Both the AFP client and the server use the National Institute of Standards and Technology Data Encryption Standard (NIST DES) algorithm to encrypt the random number. The user's case-sensitive password is applied as the encryption key to generate an eight-byte value. The server applies the same algorithm to the password it finds associated with the user name in its database.

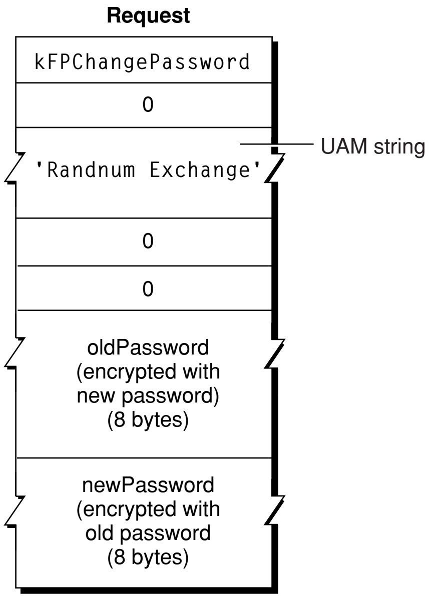
5. The AFP client sends the encrypted value to the server in the UserAuthInfo parameter of the FPLoginCont command, along with the ID number it received from the server. The server uses the ID number to associate a previous FPLogin or FPLoginExt command with its corresponding FPLoginCont command.

6. The server compares the AFP client's encrypted value with the encrypted value obtained using the password from its user database. If the two encrypted values match, the authentication process is complete and the login succeeds. The server returns a result code of kFPNoErr to the AFP client. If the two encrypted values do not match, the server returns the kFPUserNotAuth result code.

Note: The Random Number Exchange UAM uses eight-byte passwords consisting of eight-bit ASCII characters. The NIST DES algorithm ignores the low-order bit of each byte thereby using only 56 bits of the 64-bit password. The result is that in passwords, "0" is equivalent to "1", "b" is equivalent to "c", and so on.

Figure 1-6 (page 44) shows the request block for calling FPChangePassword when using the Random Number Exchange UAM.

Figure 1-6 Request block when using the Random Number Exchange UAM to change a password


Request
kFPChangePassword
0
'Randnum Exchange'
UAM string
0
0
oldPassword (encrypted with new password) (8 bytes)
newPassword (encrypted with old password (8 bytes)

Two-Way Random Number Exchange

With the Two-Way Random Number Exchange UAM, the user is authenticated to the server and the server is authenticated to the user, which guards against spoofing (that is, using a fake server to get passwords or data). This method uses the same steps as the Random Number Exchange UAM with three additional steps. The corresponding UAM string is `2-Way Randum`.

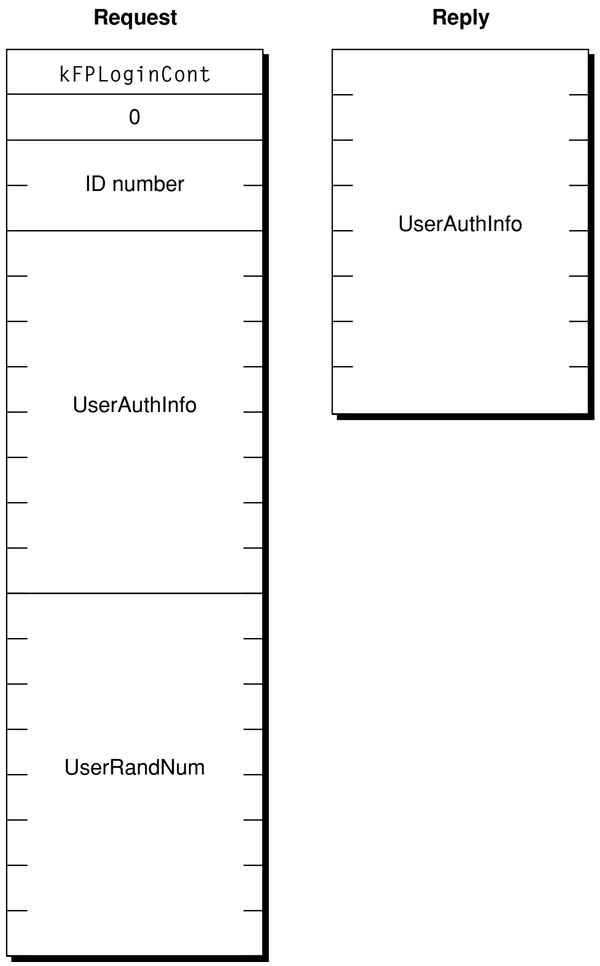
Like the Random Number Exchange UAM, the Two-Way Random Number Exchange UAM starts when the client sends the `FPLogin` or `FPLoginExt` request to the server that includes the user's user name. If the server finds the user name in the user name database, the server returns an ID number, an eight-byte random number, and a result code of `kFPAuthContinue`. The client then encodes the random number with the user's password and sends the encoded number and the ID number to the server in an `FPLoginCont` request. If the encoded password matches the server's copy of the random number encoded by the server's copy of the password, the client is authenticated and `kFPNoErr` is returned.

The additional steps of the Two-Way Random Number Exchange are

1. The client sends to the server an `FPLoginCont` request that includes a second eight-byte random number.
2. The server encodes the second eight-byte random number with it's copy of the user's password from the user database and returns the encoded random number in the `FPLoginCont` reply block.
3. The client encodes the random number with the user's password and compares it with the encoded random number from the server. If they match, the server is also authenticated.

Figure 1-7 (page 45) shows the request and reply block formats for the `FPLoginCont` command when the Two-Way Random Number Exchange UAM is used.

Figure 1-7 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. Instead, the Random Number Exchange UAM should be used to change a password. A user who has already logged in using the Two-Way Random Number Exchange UAM and who is changing his or her password has already authenticated the server, so there is no need to authenticate the server again by using the Two-Way Random Number Exchange UAM.

Note: With the Two-Way Random Number Exchange UAM, each password byte is shifted left one bit before it is used to encrypt the random number. This shifting causes the password's high-order bit to be ignored by the NIST DES algorithm instead of the low-order bit as with the Random Number Exchange UAM. 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.

Diffie-Hellman Exchange

Diffie-Hellman Exchange (DHX) is an implementation of the Diffie-Hellman Key Agreement Protocol using the SSLeay/OpenSSL implementation of CAST 128 in CBC mode. The UAM protocol name for DHX is `'DHCAST128'`.

DHX is strong against packet sniffing attacks but vulnerable to active attacks such "Man in the Middle." There is no way for the client to verify that the server knows the password, so the server could easily be spoofed. There is some weakness in using fixed initialization vectors, p and g. DHX is useful when the server requires passwords in cleartext.

With DHX, the client and the server each generate a random number, Ra and Rb respectively, which serve as "private keys" for the session. The client and server use modulus exponentiation to derive "public keys", Ma and Mb, from the private keys and exchange them. The client combines Ra and Mb, and the server combines Ma with Rb to generate identical session keys, K.

After the key exchange is complete, a key verification phase follows. Each side generates a random number (nonce), encrypts it with the session key, and sends it to the other side. Each side takes the other's verifier, decrypts to get the nonce, modifies the nonce in a way that is known to both parties, encrypts it with the session key, and sends it back. The originator verifies that the nonce was modified as expected. Incrementing the nonce is a simple and effective way of modifying the verifier.

Table 1-21 lists the values used to calculate the content of messages exchanged between the client and server when the UAM is DHX.

Table 1-15 Variables and notation used by the DHX UAM

Value	Meaning
password	User password padded with nulls to 64 bytes.
username	Pascal string (pstring), padded to an even byte length.
ServerSig	Obtained from the server by sending `FPGetSrvrInfo` command.Due to a problem in the initial implementation, the ServerSig must be set to 16 bytes of 0x00 in message #2.
AFP Vers	Pascal string (pstring) denoting the version of the AFP protocol used for the session.
ID	A two-byte number used by the server to keep track of the login/change password request. The server may send any two-byte number, the client passes it back unchanged.
x^y	Raise x to the yth power.
nonce	A random number.
nonce + 1	Add one to the nonce.
Ra	32 byte (256 bits) random number used internally by the client.
Rb	32-byte (256 bit) random number used internally by the server.

Value	Meaning
p	16 byte (128 bit) prime number satisfying the property that (p - 1)/2 is also prime (called a Sophie Germain prime).
g	A small number that is primitive mod p.
Ma	g^Ra mod p (sent by the client to the server in the first message); 16 bytes.
Mb	g^Rb mod p (sent by the server to the client in the second message); 16 bytes.
K	Key = Mb^Ra mod p = Ma^Rb mod p.
(dataBytes)K	Encrypt dataBytes using CAST 128 CBC with key, K.

For DHX, the constants p and g are defined as follows (MSB first):

```
UInt8 p = { 0xBA, 0x28, 0x73, 0xDF, 0xB0, 0x60, 0x57, 0xD4, 0x3F, 0x20, 0x24,
0x74, 0x4C, 0xEE, 0xE7, 0x5B  };
UInt8 g = { 0x07 };
```

Note: Numbers are encoded in network byte order; most significant byte (MSB) first.

Logging in Using DHX

The login sequence when using the DHX UAM consists of an exchange of the four messages shown in Table 1-16. In Table 1-16, the pipe symbol (`|`) is used to separate the elements that make up the message.

Table 1-16 Login sequence using DHX

Message	Sender/Receiver	Content
1	Client to server	| `FPLogin` (2 bytes) | AFP Vers | '`DHCAST128`' | username (padded) | Ma |
2	Server to client	| ID | Mb | (nonce, ServerSig)K | and a result code
3	Client to server	| `FPLoginCont` (2 bytes) | ID | (nonce + 1, password)K |
4	Server to client	A result code of `kFPNoErr` if authentication was successful

In response to Message 1, the server may return the following result codes (but it may delay sending some of these result codes until Message 4):

- `kFPBadUAM` — the server doesn't support the DHX UAM.
- `kFPBadVersNum` — the server doesn't support the requested AFP version.
- `kFPParamErr` — the user name is not valid.
- `kFPMiscErr` — the session is already authenticated.

- `kFPServerGoingDown` — the server is shutting down.
- `kFPUserAlreadyLoggedOnErr` — the server allows only one active session per user.
- `kFPAuthContinue` — the server is prepared to continue to login process.

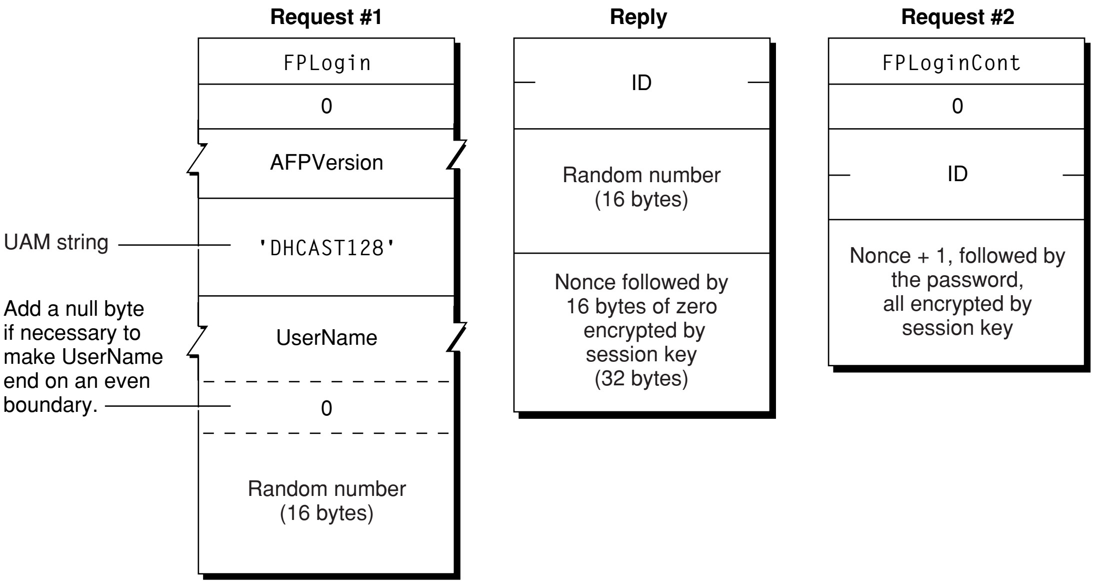
The server may delay sending some of the above result codes until the fourth message or may report a `kFPUserNotAuth` result as `kFPParamErr` to limit the amount of information disclosed to the client.

In response to Message 3, the server may return any of the following result codes:

- `kFPNoErr` — authentication was successful; the server decrypted the nonce/password and verified that the nonce was incremented properly and the password sent by the client matches the password on the server.
- `kFPUserNotAuth` — the password is incorrect.
- `kFPParamErr` — authentication failed and the server prefers not to indicate whether the user name or the password is invalid.
- `kFPPwdExpiredErr` — the user's password has expired.
- `kFPPwdNeedsChangeErr` — the user's password needs to be changed.

Figure 1-8 (page 48) shows the request and reply blocks for `FPLogin` when using the DHX UAM.

Figure 1-8 Request and reply blocks when using DHX with `FPLogin`


Request #1
FPLogin
0
AFPVersion
UAM string
'DHCAST128'
Add a null byte if necessary to make UserName end on an even boundary.
UserName
0
Random number (16 bytes)
Reply
ID
Random number (16 bytes)
Nonce followed by 16 bytes of zero encrypted by session key (32 bytes)
Request #2
FPLoginCont
0
ID
Nonce + 1, followed by the password, all encrypted by session key

Changing Passwords Using DHX

There is no equivalent to `FPLoginCont` when changing a password, so the client has send the `FPChangePassword` command at least twice and use the ID to keep track of the state of the password-changing process. The ID first appears in Message 1 and is set to 2 bytes of 0x00. The server sends a non-zero value for ID in Message 2, and the client must copy it from Message 2 into Message 3. The key used to encrypt the old and new passwords is created in the same way as the key when logging in. The values of p and g are the same values that are used when logging in.

When using the DHX UAM, the password changing sequence consists of an exchange of at least four messages shown in Table 1-17. In Table 1-17, the pipe symbol (|) is used to separate the elements that make up the message.

Table 1-17 Password-changing sequence using DHX

Message	Sender/Receiver	Content
1	Client to server	| FPChangePassword (2 bytes) | 'DHCAST128' | Username (padded) | ID (0x00 0x00) | Ma |
2	Server to client	| ID | Mb | (nonce, ServerSig)K | and a result code
3	Client to server	| FPChangePassword (2 bytes) | 'DHCAST128' | Username (padded) | ID | (nonce + 1, newPassword, oldPassword)K |
4	Server to client	A result code of kFPNoErr if the password was changed

In response to Message 1, the server may return any of the following result codes (or may wait until it receives the second FPChangePassword command to return the first three result codes):

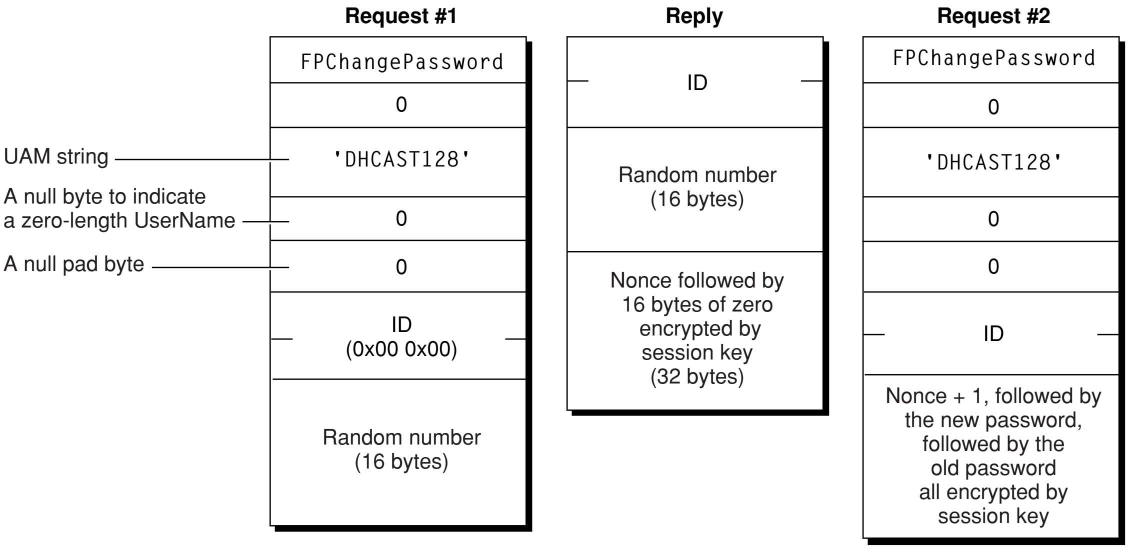
- kFPBadUAM —the server doesn't support DHX for changing passwords.
- kFPParamErr — the user name is not valid.
- kFPServerGoingDown — the server is shutting down.
- kFPAuthContinue — the server is prepared to continue the password-changing process.

In response to Message 3, the server may return any of the following result codes:

- kFPNoErr — the password was changed.
- kFPUserNotAuth — the old password is incorrect.
- kFPParamErr — to limit the amount of information released to the client.
- kFPPwdPolicyErr — the new password does not conform to the server's password policy.
- kFPPwdSameErr — the new password is the same as the old password.
- kFPPwdTooShortErr — the new password is too short.

Figure 1-9 (page 50) shows the request and reply blocks for calling FPChangePassword with the DHX UAM.

Figure 1-9 Request and reply blocks when using DHX with `FPChangePassword`



Diffie-Hellman Exchange 2

Diffie-Hellman Exchange 2 (DHX2) is an implementation of the Diffie-Hellman Key Agreement Protocol using the SSLeay/OpenSSL implementation of CAST 128 in CBC mode. The UAM protocol name for DHX2 is `'DHX2'`.

DHX2 differs from DHX in that DHX2 uses variable-sized prime (p) and generator (g) values, which allows servers to choose an appropriate level of security. The minimum size of the prime is increased to 512 bits to improve resistance to numerical methods of attack. In addition, unlike DHX, DHX2 does not use the server signature(ServerSig) in Message 2.

DHX2 is strong against packet sniffing attacks but vulnerable to active attacks such "Man in the Middle." There is no way for the client to verify that the server knows the password, so the server could easily be spoofed. There is some weakness in using fixed initialization vectors, p and g, which is alleviated by putting the random nonces first in the encrypted portions of the messages. DHX2 is useful when the server requires passwords in cleartext.

As with DHX, in DHX2 the client and server each generate a random number, Ra and Rb respectively, which serve as "private keys" for the session. The client and server use modulus exponentiation to derive "public keys", Ma and Mb, from the private keys and exchange them. The client combines Ra and Mb, and the server combines Ma with Rb to generate identical session keys, K.

After the key exchange is complete, a key verification phase follows. Each side generates a random number (nonce), encrypts it with the session key, and sends it to the other side. Each side takes the other's verifier, decrypts to get the nonce, modifies the nonce in a way that is known to both parties, encrypts it with the session key, and sends it back. The originator verifies that the nonce was modified as expected. Incrementing the nonce is a simple and effective way of modifying the verifier.

Table 1-21 lists the values used to calculate the content of messages exchanged between the client and server when the UAM is DHX2.

Table 1-18 Variables used by the DHX2 UAM

Value	Meaning
password	User password padded with nulls to 64 bytes.
username	Pascal string (pstring), padded to an even byte length.
AFP Vers	Pascal string (pstring) denoting the version of the AFP protocol used for the session.
ID	A two-byte number used by the server to keep track of the login/change password request. The server may send any two-byte number, the client passes it back unchanged.
ID + 1	The ID incremented by one.
clientNonce	A 16-byte random number used in the key verification portion of the exchange.
serverNonce	A 16-byte random number used in the key verification portion of the exchange.
clientNonce + 1	The clientNonce incremented by one.
MD5(data)	Take the MD5 hash of the data, which results in a 16-byte (128 bit) value.
p	A variable length prime number (at minimum 512 bits in size) satisfying the property that (p - 1)/2 is also prime (called a Sophie Germain prime) sent by the server to the client. (Two bytes length followed by data)
g	A small number that is primitive mod p sent by the server to the client. (Four bytes)
x^y	Raise x to the yth power.
Ra	An x bit random number used internally by the client.
Rb	An x bit random number used internally by the server.
Ma	g^Ra mod p (sent by the client to the server); the same number of bytes as p, padded with nulls at the MSB end.
Mb	g^Rb mod p (sent by the server to the client); the same number of bytes as p, padded with nulls at the MSB end.
x	The size of p in bits.
len	The size of p & Ma & Mb in bytes; a two-byte value.
K	Key = MD5(Mb^Ra mod p) = MD5(Ma^Rb mod p)
(dataBytes)K	Encrypt dataBytes using CAST 128 CBC with key, K.

Note: Numbers are encoded in network byte order; most significant byte (MSB) first.

Logging In Using DHX2

When using the DHX2 UAM, the login sequence consists of an exchange of the six messages shown in Table 1-19. In Table 1-19, the pipe symbol (|) is used to separate the elements that make up the message.

Table 1-19 Login sequence using DHX2

Message	Sender/Receiver	Content
1	Client to server	| `FPLogin` (2 bytes) | AFP Vers | ‘`DHX2`’ | Username (padded) |
2	Server to client	| ID | g | len | p | Mb | and a result code
3	Client to server	| `FPLoginCont` (2 bytes) | ID | Ma | (client nonce)K |
4	Server to client	| ID + 1 | (clientNonce + 1, serverNonce)K | and a result code
5	Client to server	| `FPLoginCont` (2 bytes) | ID + 1 | (serverNonce+1, password)K |
6	Server to client	A result code of `kFPNoErr` if authentication was successful

In response to Message 1, the server may return the following result codes (but it may delay sending some of these result codes until Message 6):

- `kFPBadUAM` — the server doesn’t support the DHX2 UAM.
- `kFPBadVersNum` — the server doesn't support the requested AFP version.
- `kFPParamErr` — the user name is not valid.
- `kFPMiscErr` — the session is already authenticated.
- `kFPServerGoingDown` — the server is shutting down.
- `kFPUserAlreadyLoggedOnErr` — the server allows only one active session per user.
- `kFPAuthContinue` — the server is prepared to continue to login process.

The server may delay sending some of the above result codes until the sixth message or may report a `kFPUserNotAuth` result as `kFPParamErr` to limit the amount of information disclosed to the client.

In response to the `FPLoginCont` command, the server may return any of the following result codes:

- `kFPNoErr` — authentication was successful; the server decrypted the nonce/password and verified that the nonce was incremented properly and the password sent by the client matches the password on the server
- `kFPUserNotAuth` — the password is incorrect
- `kFPParamErr` — authentication failed and the server prefers not to indicate whether the user name or the password is invalid
- `kFPPwdExpiredErr` — the user’s password has expired

- kFPPwdNeedsChangeErr — the user's password needs to be changed

Changing Passwords Using DHX2

There is no equivalent to FPLoginCont when changing a password, so the client has send the FPChangePassword command at least twice and use the ID to keep track of the state of the password-changing process. The ID first appears in Message 1 and is set to 2 bytes of 0x00. The server sends a non-zero value for ID in Message 2, and the client must copy it from Message 2 into Message 3 as well as from Message 4 into Message 5. The key used to encrypt the old and new passwords is created in the same way as the key when logging in. The values of p and g are the same values that are used when logging in.

When using the DHX2 UAM, the password changing sequence consists of an exchange of at least six messages shown in Table 1-20. In Table 1-20, the pipe symbol (|) is used to separate the elements that make up the message.

Table 1-20 Password-changing sequence using DHX2

Message	Sender/Receiver	Content
1	Client to server	| FPChangePassword (2 bytes) | 'DHX2' | Username (padded) | ID (0x00 0x00) |
2	Server to client	| ID | g | len | p | Mb | and a result code
3	Client to server	| FPChangePassword (2 bytes) | 'DHX2' | Username (padded) | ID | Ma | (clientNonce)K |
4	Server to client	| ID+1 | (clientNonce+1, serverNonce)K | and a result code
5	Client to server	| FPChangePassword (2 bytes) | 'DHX2' | Username (padded) | ID+1 | (serverNonce+1, newPassword, oldPassword)K |
6	Server to client	A result code of kFPNoErr if the password was changed

In response to Message 1, the server may return kFPAuthContinue or any of the following result codes:

- kFPBadUAM —the server doesn't support DHX2 for changing passwords.
- kFPParamErr — the user name is not valid.
- kFPServerGoingDown — the server is shutting down.

In response to Message 3, the server may return kFPAuthContinue or any of the following result codes:

- kFPUserNotAuth — the old password is incorrect.
- kFPPwdPolicyErr — the new password does not conform to the server's password policy.
- kFPPwdSameErr — the new password is the same as the old password.
- kFPPwdTooShortErr — the new password is too short.

Kerberos

The AFP client learns whether a server supports the Kerberos UAM by examining the `kSupportsDirServices` bit in the `Flags` parameter returned by the `FPGetSrvrInfo` command. If that bit is set, a server that supports Kerberos UAM places its principal name in the `DirectoryNames` parameter returned by `FPGetSrvrInfo`.

The AFP client uses the principal name to determine if the server supports Kerberos v4 or v5.

Note: Mac OS X AFP servers only support Kerberos V5 authentication.

Then the client tries to get a service ticket from the server. If it cannot get a ticket, the client must use some other authentication method. If the client gets a service ticket, it can call `FPLoginExt`, providing the following values in the request block:

- two-byte `Flags` parameter
- AFP Version string
- UAM string (`Client Krb v2`)
- kFPUT8Name
- length of the user name in Unicode characters that follows
- user name in Unicode characters
- kFPUT8Name
- length of the realm in Unicode characters that follows
- realm in Unicode characters

The server replies with a result code of `kFPAuthContinue`. The reply block contains a two-byte `ID` value.

If the client is using Kerberos v4, it calls `FPLoginCont`, providing the following values in the request block:

- user name in Unicode characters
- pad byte if one is necessary to force user name to end on an even boundary
- length of the ticket that follows
- ticket, created by `KClientGetTicketForService()`

The user is authenticated if the server returns a result code of `kFPNoErr` and a reply block consisting of a two-byte length parameter and an authenticator.

If the client is using Kerberos v5, it calls `FPLoginCont`, providing the following values in the request block:

- ID returned by `FPLoginExt`
- user name in Unicode characters
- pad byte if one is necessary to force user name to end on an even boundary

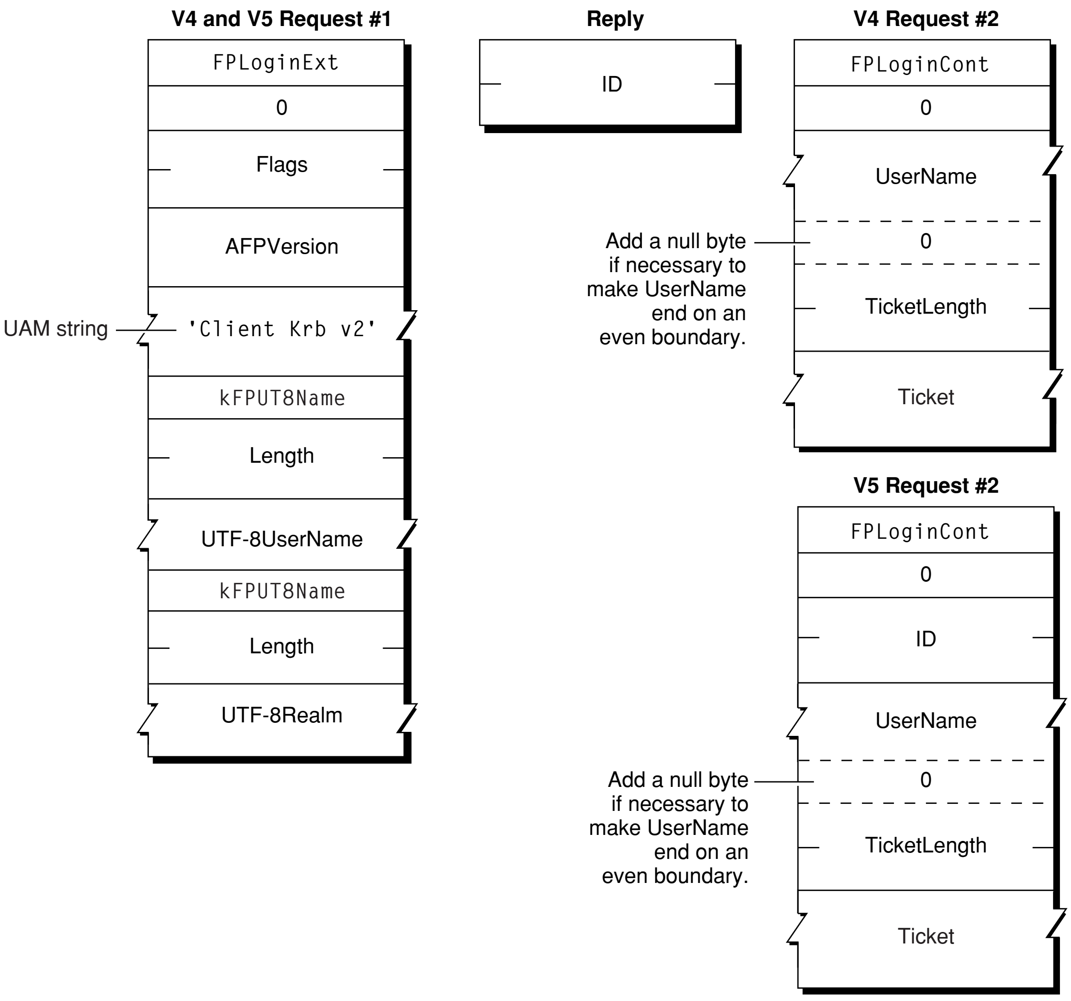
- length of the ticket that follows
- ticket, created by `gss_init_sec_context` with `GSS_C_MUTUAL_FLAG` and `GSS_C_REPLAY_FLAG` set and no channel bindings

The user is authenticated if the server returns a result code of `kFPNoErr` and a reply block consisting of a two-byte length parameter and an authenticator.

After the client receives the `FPLoginCont` reply packet, the client sends an `FPGetSessionToken` command with a type of `kGetKerberosSessionKey` (8) in order to get a random session key from the server. This session key is encrypted on the server using `gss_wrap()` and is decrypted on the client using `gss_unwrap()`. Note that the client may call `FPGetSessionToken` later on in order to get a disconnect token.

Figure 1-10 (page 55) shows the request and reply blocks for `FPLoginExt` and `FPLoginCont` when using the Kerberos UAM.

Figure 1-10 Request and reply blocks when using Kerberos with `FPLoginExt`



Reconnect

Unlike the other UAMs described in this section, which are used to log in to an AFP server, the Reconnect UAM is used only to reconnect to a server. The Reconnect UAM can be used when the original connection was made using a UAM that provides a session key, such as DHX, DHX2, and Kerberos. The UAM protocol name for the Reconnect UAM is 'Recon1'.

The goals of the Reconnect UAM are:

- Store in a token returned by the FPGetSessionToken command all of the information required to reconnect, even if the server has been rebooted.
- Use only a secure hash function and a symmetric encryption algorithm.
- Provide mutual authentication to prove that the server to which the client is reconnecting is the same server that was originally authenticated.
- Ensure that a compromised session key or seed value will not compromise the long term server key.

Table 1-21 lists the variables used to calculate values for the Reconnect UAM.

Table 1-21 Variables used by the Reconnect UAM

Value	Size in Bytes	Meaning
k1	16	Initial session key returned by the UAM that was used to log in; known to both the client and the server at the time of reconnect.
ks	16	Long term server key.
s	8	Lamports hash seed.
n	4	Number of times to run the hash function.
m	4	Maximum number of times to run the hash function (m >= n).
clientNonce	8	A random number selected by the client nonce.
serverNonce	8	A random number selected by the server.
t1	4	Initial timestamp.
t2	4	Timestamp used when reconnecting.
t3	4	Time interval between the server's clock and the client's clock.
exp	4	Credential's expiration time.
now	4	Current time as known by the server or by the client.

Value	Size in Bytes	Meaning
user/domain		Username information that uniquely identifies the user.
sessionInfo		Information that uniquely identifies a session.
(data)key		Data encrypted using a symmetric encryption algorithm using key. (CBC mode)
(data)H		Data hashed with a secure hash function.
(data)H(n)		Data hashed n times with a secure hash function.
(data)HMAC(key)		Data signed by a keyed HMAC algorithm.
revocation list		List of hash value and time-to-live pairs. Pairs stay in the list until the time-to-live value has passed.
cred		(s, m, exp, t3, user/domain)ks

Table 1-22 describes common methods of attack and the ways in which the Reconnect UAM is protected from these attacks.

Table 1-22 Attacks on the Reconnect UAM

Attack	Defense
Man in the Middle	If the original UAM used to connect to the server was resistant to Man in the Middle attacks, nonce checks in message *a*, which require knowledge of *s*, should keep out the Man in the Middle.
Replay	The timestamp in message *a*, protected by the HMAC, and the credential revocation list should prevent simple replay attacks. Even if the attacker succeeds in controlling the clock on the server and manages to force a server restart, the attacker cannot log in because s is not known, so the challenge/response step cannot be performed successfully.
Reflection	This type of attack is thwarted by the use of chained nonces, by having the user information in the credential, and by having each message be non-symmetrical.
Interleaving	This type of attack is thwarted by the use of chained nonces.
Chosen Text	The server's key is not used to encrypt any data that is obtained from the client.
Forced Delay	Timestamps, key expiration and the use of the revocation list should thwart this type of attack.

Getting a Credential

After the client successfully logs in and mounts a remote volume, it calls FPGetSessionToken, setting the Type parameter to kRecon1Login (5), and sending to the server its initial timestamp (t1) encrypted with the session key (k1):

(t1)k1

As a result of the login process, the server also knows the session key (k1) and the sessionInfo for this session. The server also has a long term session key (ks).

The server uses t1 to compute the clock skew and determine an appropriate expiration time for the credential it is about to create. The server then generates a credential by concatenating the Lamports hash seed (s), the maximum number of times to run the hash function (m), the expiration time, the user/domain, and encrypting the concatenation using its long term session key (ks):

cred = (s, m, exp, t3, user/domain)ks

The server also computes (cred)H and stores the result in its revocation list. The server then uses the session key (k1) to encrypt a concatenation of the credential (cred), the Lamports hash seed (s), the maximum number of times to run the hash function (m), the expiration time (exp), and sessionInfo, and sends the result to the client. The formula for this calculation is:

(cred, s, m, exp, sessionInfo)k1

The client uses the session key (k1) to decrypt the result, obtaining the encrypted credential, the Lamports hash seed, the maximum number of times to run the hash function, the encrypted credential's expiration time, and the sessionInfo. The client is responsible for storing this information so that it can use it later.

Table 1-23 summarizes the exchange between client and server when getting a credential.

Table 1-23 Getting a credential

Message	Sender/Receiver	Content
1	Client to server	| FPGetSessionToken (2 bytes) | kRecon1Login | IDLength | (t1)k1 | ID |
2	Server to client	| (cred, s, m, exp, sessionInfo)k1 |

Refreshing the Credential

Before the credential expires, the client calls FPGetSessionToken again, setting the Type parameter to kRecon1RefreshToken (7) and sending to the server the initial timestamp (t1) and the current credential encrypted with the session key (k1). The formula for this calculating this value is:

(t1, cred)k1

Both the client and the server compute k2 using the following formula:

k2 = (cred, s)H

The server decrypts the value sent by the client. If the credential is valid, the server creates a new credential encrypted with the long term session key and a new expiration time, stores (cred')H on the revocation list, and returns the encrypted credential to the client along with a new Lamports hash seed, a new maximum number of times to run the hash function, and the sessionInfo, all encrypted by k2. The formula for creating this value is:

(cred', s', m', exp', sessionInfo)k2

The client uses k2 to decrypt the reply, obtaining the new credential, the new Lamports hash seed, the new maximum number of times to run the hash function, the new expiration and the sessionInfo. Before this credential expires, the client refreshes it again.

Table 1-24 summarizes the exchange between client and server when refreshing a credential.

Table 1-24 Refreshing a credential

Message	Sender/Receiver	Content
1	Client to server	| FPGetSessionToken (2 bytes) | kRecon1RefreshToken | IDLength | (t1, cred)k1 | ID |
2	Server to client	| (cred', s', m', exp', sessionInfo)k2 |

Using the Credential to Reconnect

If the connection to the server goes down for any reason, the client has the current credential, the Lamports hash seed (s), and the maximum number of times to run the hash (m).

The client logs back in using the FPLoginExt command, specifying Recon1ReconnectLogin as the UAM, and sending the following information to the server:

(cred, (s)H(n), n, t2, (clientNonce)[(s)H(n-1])HMAC(s)

The server uses its long term session key (ks) to decrypt cred. If decryption fails, the server fails the login attempt. It also retrieves s, m, exp, t3, and user/domain.

If the decryption succeeds, the server computes (cred)H and looks it up in the revocation list. If found, the credential has expired, so the server fails the login attempt.

If (cred)H is not found in the revocation list, the server checks exp, m >= n, and HMAC(s) user/domain. If any are invalid, the server fails the login attempt.

The server then computes and compares (s)H(n)' and (s)H(n1). If they don't match, the server fails the login attempt.

The server then decrypts and hashes clientNonce, chooses serverNonce, adds (cred)H, t3+now to the revocation list, and sends the following value to the client:

(serverNonce, (clientNonce)H)[(s)H(n-1)]

The client decrypts the value, verifies (clientNonce)H, and hashes serverNonce. The client uses the FPLoginCont command to send the following value to the server:

((serverNonce)H[(s)H(n-1)]

The server decrypts the value and verifies (serverNonce)H. If they don't match, the server fails the login attempt. If they match, the server replies to the client with a result code of kFPNoErr. The client is now logged in. Both the server and the client make the following calculation:

k1' = (clientNonce, serverNonce)H

The client calls FPGetSessionToken using k1' as the session key to get a new credential. It also calls FPDisconnectOldSession to tell the server to disconnect the old session and transfer is resources to the new session.

Table 1-25 summarizes the exchange between client and server when reconnecting.

Table 1-25 Reconnecting using the Recon1 UAM

Message	Sender/Receiver	Content
1	Client to server	| FPLoginExt (2 bytes) | Flags | AFP version | 'Recon1' | UserNameType | UserName | PathType | Pathname | (cred, (s)H(n), n, t2, (clientNonce)[(s)H(n-1])HMAC(s) |
2	Server to client	|(serverNonce, (clientNonce)H)[(s)H(n-1)] | and a result code
3	Client to server	| FPLoginCont (2 bytes) | ID | ((serverNonce)H[(s)H(n-1)] |
4	Server to client	kFPNoErr or another result code indicating log in failure
5	Client to server if result is kFPNoErr	| FPGetSessionToken (2 bytes) | kRecon1ReconnectLogin | IDLength| (t1, cred)k1 | ID |

Volume Passwords

AFP provides an optional second-level of access control through volume passwords. A server can associate a fixed-length 8-character password with each volume it makes visible to AFP clients.

The AFP client can issue an FPGetSrvrParms command to the server to discover the names of each volume and to get an indication of whether each of them is password-protected.

To send AFP commands that refer to a server volume, the AFP client uses a volume identifier called the Volume ID. The AFP client obtains this ID by sending an FPOpenVol command to the server. This command contains the name of the volume as one of its parameters. If a password is associated with the volume, the command must also include the password as another parameter.

Volume passwords constitute a simple protection for servers that do not need to implement the directory access controls described in the next section. However, volume passwords are not as secure as directory access controls.

Directory Access Controls

Directory access controls provide the greatest degree of network security in AFP by access privileges to users. Once the user has logged in, access privileges allow users varying degrees of freedom for performing actions within the directory structure.

AFP defines three directory access privileges: search, read, and write:

- A user with *search* access to a directory can list the parameters of directories contained within the directory.
- A user with *read* access to a directory can list the parameters of files contained within the directory in addition to being able to read the contents of a file.
- A user with *write* access to a directory can modify the contents of a directory including the parameters of files and directories contained within the directory. Write access allows the user to ad and delete directories and files as well as modify the data contained within a file.

Each directory on a server volume has an owner and a group affiliation. Initially, the owner is the user who created the directory, although ownership of a directory may be transferred to another user. Only the owner of a directory can change its access privileges. The server uses a name of up to 31 characters and a four-byte ID number to represent owners of directories. Owner name and Owner ID are synonymous with User name and User ID.

The group affiliation is used to assign a different set of access privileges for the directory to a group of users. For each group, the server maintains a name of up to 31 characters, a four-byte ID number and a list of users belonging to the group. Assigning group access privileges to a directory gives those privileges to that set of users.

Each user may belong to any number of groups or to no group. One of the user's group affiliations may be designated as the user's primary group. This group will be assigned initially to each new directory created by the user. The directory's group affiliation may be removed or changed later by the owner of the directory.

The term *Everyone* is used to indicate every user that is able to log in to the server. A directory may be assigned certain access privileges for Everyone that would be granted to a user who is neither the directory's owner nor a member of the group with which the directory is affiliated.

With each directory, the file server stores three access privileges bytes, which correspond to the owner of the directory, its group affiliation, and Everyone. Each of these bytes is a bitmap that encodes the access privileges (search, read, and write) that correspond to each category. The most significant bits of each access privileges byte must be zero.

To perform directory access control, AFP associates the five parameters shown in Table 1-26 with each directory.

Table 1-26 Directory access control parameters

Parameter	Size
Owner ID	Four bytes
Group ID	Four bytes

Parameter	Size
Owner access privileges	One byte
Group access privileges	One byte
Everyone access privileges	One byte

The Owner ID is the same as the owner’s User ID. The Group ID is the ID number of the group with which the directory is affiliated, or zero. The file server maintains a one-to-one mapping between the Owner ID and the user name and between the Group ID and the group name. As a result, each name is associated with a unique ID. AFP includes commands that allow users to map IDs to names and names to IDs. Assignment of User IDs, Group IDs, and primary groups is an administrative function and is outside the scope of this protocol.

A Group ID of zero means that the directory has no group affiliation. The groups access privileges are ignored.

When a user logs on to a server, identifiers are retrieved from a user database maintained on the server. These identifiers include the User ID (a four-byte number unique among all server users) and one or more four-byte Group IDs, which indicate the user’s group memberships. The exact number of group memberships is implementation-dependent. One of these Group IDs may represent the user’s primary group.

The server must be able to derive what access privileges a particular user has to a certain directory. The user access privileges (UARights) contain a summary of the privileges, regardless of the category (Owner, Group, Everyone) from which they were obtained. In addition, the user access privileges contain a flag indicating whether the user owns the directory.

The server uses the following algorithm to extract user access privileges. The OR in this algorithm indicates inclusive OR operations.

```
UARights := Everyone’s access rights;
clear UARights owner flag
If (Owner ID = 0) then
    set UARights own flag
If (User ID = Owner ID) then
    UARights := UARights OR owner’s access privileges;
    set UARights owner flag
If (any of user’s Group IDs = directory’s Group ID) then
    UARights := UARights OR directory;s group access privileges
```

An Owner ID of zero means that the directory is not owned or is owned by another user. The owner bit of the access privileges byte is always set for such a directory.

The access privileges required by the user to perform most file management functions are explained in the following paragraphs according to the symbols listed in Table 1-27.

Table 1-27 Access privilege notation

Symbol	Meaning
SA	Search access to all ancestors down to, but not including the parent directory
WA	Search or write access to all ancestors down to, but not including, the parent directory
SP	Search access to the parent directory
RP	Read access to the parent directory
WP	Write access to the parent directory

Almost all operations require *SA*. To perform any action within a given directory, the user must have permission to search every directory in the path from the root to the parent's parent directory. Access to files and directories within the parent directory is then determined by *SP*, *RP*, and *WP*.

Specific file management functions and the access privileges needed to perform them are listed in Table 1-28.

Table 1-28 File management functions and required privileges

Function	Required access privileges
Create a file or a directory	The user must have *WA* plus *WP*. A hard create (delete first of the file exists) requires the same privileges as deleting a file.
Enumerate a directory	To enumerate a directory is to list in numerical order the offspring of the directory and selected parameters of those offspring. The user must have search access to all directories down to but not necessarily including the directory being enumerated (*SA*). In addition, to view its directory offspring, the user must have search access to the directory being enumerated (*SP*). To view its file offspring, search access to the directory is not required, but the user must have read access to the directory (*RP*).
Delete a file	The user must have *SA*, *RP*, and *WP*. A file can be deleted only it if is not open at that time.
Delete a directory	The user must have *WA* plus *WP*. A hard create (delete first of the file exists) requires the same privileges as deleting a file.
Rename a file	To enumerate a directory is to list in numerical order the offspring of the directory and selected parameters of those offspring. The user must have search access to all directories down to but not necessarily including the directory being enumerated (*SA*). In addition, to view its directory offspring, the user must have search access to the directory being enumerated (*SP*). To view its file offspring, search access to the directory is not required, but the user must have read access to the directory (*RP*).

Function	Required access privileges
Rename a directory	The user must have *SA*, *RP*, and *WP*. A file can be deleted only it if is not open at that time.
Rename a file	The user must have *SA*, *SP*, and *WP*. A directory can be deleted only if it is empty
Rename a directory	The user must have *SA*, *RP*, and *WP*.
Read directory parameters	The user must have SA and SP.
Open a file for reading	A file's fork must be opened in read mode before its contents can be read. To open a file in read mode, the user must have *SA* and *RP*. Read mode and other access modes are described in the next section.
Open a file for writing	A file's fork must be opened in write mode in order to write to it. To open an empty fork for writing, the user must have *WA* and *WP*. (The empty fork must belong to a file that has both forks of zero length. To open an existing fork (when either fork is not empty) for writing, *SA*, *RP*, and *WP* are required.
Write file parameters	For an empty file (where both forks are zero length), the user must have *WA* plus *WP*. For a non-empty file (where one or both forks are not zero length), the user must have *SA*, *RP*, and *WP*.
Write directory parameters	For directories that contain offspring, the user must *SA*, *SP*, and *WP*. For directories that are empty, the user must have *WA* and *WP*.
Move a directory or a file	Through AFP, a directory or a file can be moved from its parent directory to a destination parent directory on the same volume. To move a directory, the use must have *SA* and *SP* to the source parent directory, *WA* to the destination parent directory, plus *WA* to both the source and the destination parent directories. To move a file, the user needs *SA* plus *RP* to the source parent directory, plus *WP* to both the source and the destination parent directories.
Modify a directory's privileges	A directory's Owner ID, Group ID, and the three access privileges bytes can be modified only if the user is the directory's owner and then only if the user has *WA* plus *WP* or *SP* access to the parent directory.
Copy a file (`FPCopyFile`)	To copy a file, on a single volume or across volumes managed by the server, the user must have *SA* plus *RP* access to the source parent and *WA* plus *WP* to the destination parent directory.

Inherited Access Privileges

AFP Version 2.1 and later supports inherited access privileges through the directory's Blank Access Privileges bit in the Directory bitmap. When the Blank Access Privileges bit is set for a directory, its other access privilege bits are ignored and the access privilege bits of the directory's parent apply to the directory, including the parent's group affiliation.

The Blank Access Privileges bit cannot be set for a directory that is a share point. Likewise, the Blank Access Privileges bit cannot be set for a volume root directory (Directory ID = 2) of a shared volume because it is always a share point for the administrator/owner.

Important
Inherited access privileges are useful because they cause access privileges to behave as users expect them to: When a directory with the Blank Access Privileges bit set is moved within the directory hierarchy, it always reflects the access privileges of the directory containing it. When the Blank Access Privileges bit is cleared, its current access privileges "stick" to that directory and remain unchanged no matter where the directory is moved. Therefore, although implementing inherited access privileges is optional, it is highly recommended that you include this feature in your AFP implementation as it has subtle human interface repercussions.

File Sharing Modes

AFP controls user access to shared files in two ways. The first, described in the previous section, provides security by controlling user access to specific directories. The second, described in this section, preserves data integrity by controlling a user's access to a file while it is being used by another user.

To control simultaneous file access, the file server must enforce synchronization rules. These rules prevent applications from damaging each other's files by modifying the same version simultaneously. These rules also prevent users from obtaining access to information while it is being changed.

Synchronization rules are built from the mode in which a first user and subsequent users open a file. AFP provides two classes of modes: access modes and deny modes.

Access and Deny Modes

Most file systems use a set of permissions to regulate the opening of files. This set includes permission to modify the contents of a file (read-write) and permission to see the file's contents (read only). In a stand-alone system, these two file-access modes are sufficient.

In the shared environment of a file server, this set of permissions, or access modes, is expanded. In addition to the expanded set of access modes, a set of restrictions is provided by deny modes.

A user application can specify an access mode and a deny mode when it opens a file on the file server. AFP supports the access modes: read, write, read-write, and none. None access allows no further access to the fork, except to close it, and may be useful in implementing synchronization. In addition to one of these access modes, the user indicates a deny mode to the server to specify which rights should be denied to others trying to open the fork while the first user has it open. Users that subsequently try to open that fork can be denied read, write, read-write, or none access.

A user sending an `FPOpenFork` command can be denied file access for the following reasons:

- The user does not possess the rights (as owner, group, or Everyone) to open the file with the requested access mode. A result code of `kFPAccessDenied` is returned.
- The fork is already open with a deny mode that prohibits the second user's requested access. For example, the first user opened the fork with a deny mode of DenyWrite, and the second user tries to open the for in the write mode. A `kFPDenyConflict` result code is returned to the second user.
- The fork is already open with an access mode that conflicts with the second user's requested deny mode. For example, the first user opened the fork for Write access and a deny mode of DenyNone. The second user tries to open the fork with a deny mode indicating DenyWrite. This request is not granted because the fork is already open for Write access. A `kFPDenyConflict` result code is returned to the second user.

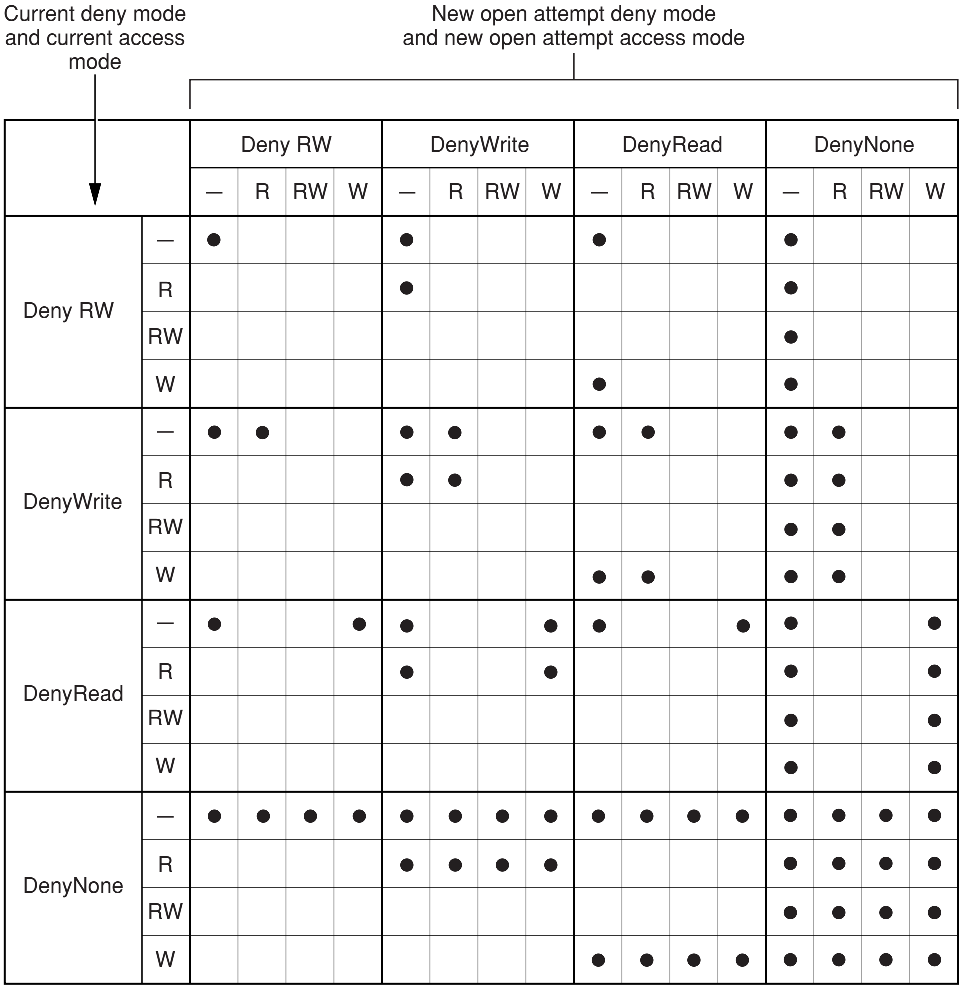
Deny modes are cumulative in that each successful opening of a fork combines its deny mode with previous deny modes. Therefore, if the first user opening a file specifies a deny mode of DenyRead, and the second user specifies DenyWrite, the fork's current deny mode is DenyRead-Write. DenyNone and DenyRead combine to form a current deny mode of DenyRead.

Similarly, access modes are cumulative. If the first user opening a file has Read access and the second has Write access, the current access mode is Read-Write.

Synchronization Rules

Synchronization rules, as previously discussed, allow or deny simultaneous access to a file fork. They are based on the current deny mode and current access mode of the fork and on the new deny and access modes being requested in a new `FPOpenFork` command. Synchronization rules are summarized in Figure 1-11 (page 67). A dot indicates that a new open command has succeeded; otherwise, it has failed.

Figure 1-11 Synchronization rules



Current deny mode and current access mode

New open attempt deny mode and new open attempt access mode

		Deny RW				DenyWrite				DenyRead				DenyNone			
		—	R	RW	W	—	R	RW	W	—	R	RW	W	—	R	RW	W
Deny RW	—	●				●				●				●			
	R					●								●			
	RW													●			
	W									●				●			
DenyWrite	—	●	●			●	●			●	●			●	●		
	R					●	●							●	●		
	RW													●	●		
	W									●	●			●	●		
DenyRead	—	●			●	●			●	●			●	●			●
	R					●			●					●			●
	RW													●			●
	W													●			●
DenyNone	—	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	R					●	●	●	●					●	●	●	●
	RW													●	●	●	●
	W									●	●	●	●	●	●	●	●

Desktop Database

For file server volumes, AFP provides an interface that replaces the Finder’s direct use of the Desktop file. This interface is necessary because the Desktop file was designed for a standalone environment and could not be shared by multiple users. The AFP interface to the Desktop database replaces the Desktop file and can be used transparently for both local and remote volumes.

The Desktop database is used by the file server to hold information needed specifically by the Finder to build its unique user interface, in which icons are used to represent objects on a disk volume. To create certain parts of this interface, the Finder uses the Desktop database to perform three functions:

- to associate documents and applications with particular icons and store the icon bitmaps
- to locate the corresponding application when a user opens a document
- to hold text comments associated with files and directories

Macintosh applications usually contain an icon that is to be displayed for the application itself as well as other icons to be displayed for documents that the application creates. These icons are stored in the application's resource fork and in the Desktop database. The Desktop database associates these icons with each file's creator (the `fdCreator` field in the FInfo record) and the type (the `fdType` field in the FInfo record), which are stored as part of the file's Finder information.

The Finder allows a Mac OS user to open a document, that is, to select a file and implicitly start the application that created the file. To do this, the Desktop database maintains a mapping between the file creator and a list of the locations of each application that has that file creator associated with it. This mapping is referred to as an APPL mapping because all Macintosh applications have a file creator of `'APPL'`. The Finder obtains the first item in the list and tries to start the application. If for some reason the application cannot be started (for example, if it is currently in use), the Finder will obtain the next application from the Desktop database's list and try that one. This list is dynamically filtered to present to the Finder only those applications for which the AFP client has the proper access rights.

The Desktop database is also a repository for the text of comments associated with files and directories on the volume. The Finder will make calls to the Desktop database to read or write these comments, which can be viewed and modified by selecting the Get Info item in the Finder's File menu. Comments are completely uninterpreted by the Desktop database.

For more information about the Finder and the use of the Desktop file, refer to *Inside Mac OS X*.

Character Encoding

If the server and the sharepoint support UTF8 names, the AFP server and client send and receive decomposed UTF8. However, characters in the range of U2000 to U2FFF, UFE30 to UFE4F, and U2F800 to U2FA1F are not decomposed. For complex characters, Unicode 3.2-based tables are used. For additional information, see `http://developer.apple.com/technotes/tn/tn1150.html#UnicodeSubtleties` and the Unicode specifications.

For Macintosh Roman, AFP utilizes character string entity names that can be composed of any 8-bit character. Character representations are exactly the same as those used by the Mac OS and are shown in Figure 1-12 (page 69).

Note: The information in this section applies only to Macintosh Roman character representations and does not apply to Unicode character representations.

Figure 1-12 AFP character set mapping

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SPACE	0	@	P	‘	p				∞		—		
1	SOH	DC1	!	1	A	Q	a	q			°	±	`			
2	STX	DC2	"	2	B	R	b	r			¢	≤	´			
3	ETX	DC3	#	3	C	S	c	s			£	≥	√			
4	EOT	DC4	$	4	D	T	d	t			/	·	˜			
5	ENQ	NAK	%	5	E	U	e	u			•	µ	≈			
6	ACK	SYN	&	6	F	V	f	v			ƒ	δ	Δ	÷		
7	BEL	ETB	’	7	G	W	g	w			§	Σ	˙	◊		
8	BS	CAN	(	8	H	X	h	x			¤	Π	¨			
9	HT	EM	)	9	I	Y	i	y			'	π				
A	LF	SUB	*	:	J	Z	j	z			“	∫	˽			
B	VT	ESC	+	;	K	[	k	{			“	”	¸			
C	FF	FS	,	<	L	\	l	|			‹	…	Ã			
D	CR	GS	-	=	M	]	m	}			≠	Ω	Õ			
E	SO	RS	.	>	N	^	n	~			fi		˛			
F	SI	US	/	?	O	_	o	DLE			fl	¿	ˇ			

Throughout AFP, character string comparison is done in a case-insensitive manner (that is, K = k), and it must also be done in a diacritical-sensitive manner (that is, e é).

The mapping in Figure 1-12 shows the rules for uppercase equivalence of characters in AFP. Any character that does not appear in this table has no uppercase equivalent in AFP and therefore can only match itself. Note that this mapping does not exactly conform to the standards used in all human languages. In certain languages, the uppercase equivalent of e is E; in other languages (and in AFP), it is E´.

Tasks

This section provides an overview of AFP commands and how they are used. Each command obtains access to an AFP-file-system-visible entity. This section groups the commands in relation to the entity that the commands address. These groups include server, volume, directory, file, combined directory-file, fork, and Desktop database commands.

Login Commands

An AFP client uses the following commands to get information about a file server and to open and close a session with it:

- `FPGetSrvrInfo`
- `FPGetAuthMethods`
- `FPLogin` and `FPLoginExt`
- `FPLoginCont`
- `FPGetSrvrParms`
- `FPGetSessionToken`
- `FPDisconnectOldSession`
- `FPLogout`
- `FPMapID`
- `FPMapName`
- `FPChangePassword`
- `FPGetUserInfo`

The AFP client sends the `FPGetSrvrInfo` command to obtain server information. The `FPGetSrvrInfo` command returns server information including the following server parameters: server name, machine type, AFP version strings, UAM strings, volume icon and mask, a bitmap of flags, and optionally, a list of available Open Directory names. For descriptions of server parameters, see `FPGetSrvrInfo` in the Reference section.

From the lists of AFP versions and UAMs that the server supports, the AFP client selects the highest AFP version and the most secure UAM that the AFP client supports. To establish a session with the file server, the AFP client includes the strings for the selected AFP version and UAM in an `FPLogin` or `FPLoginExt` command.

When calling `FPLoginExt`, the AFP client must specify the user name in Unicode format and specify the Open Directory domain in which the user can be found. (A user name specified in Unicode format is the same as a Unicode file name, except that there is no text encoding hint.) Before calling `FPLoginExt`, the AFP client may first send an `FPGetAuthMethods` command to get the authentication methods that the Directory Service domain supports.

In response to the `FPLogin` or `FPLoginExt` command, the server performs user authentication. Depending on the selected UAM, the entire user authentication process can involve one or more `FPLoginCont` commands to complete the authentication process with the server. A session is established between the file server and the AFP client when the authentication process completes successfully.

After a session is established, the AFP client must obtain a list of the server's volumes. To obtain the list, the AFP client makes an `FPGetSrvrParms` command, which returns the number of volumes shared by the server, the names of the volumes, and whether they are password-protected.

The `FPGetSessionToken` command gets a reconnect token that the AFP client may later use if the session is disconnected unintentionally. In the case of an unintentional disconnection, the AFP client logs in again using the same user and authentication information that it used to log in previously, re-establishes the state of the connection, and sends an `FPDisconnectOldSession` command that passes the reconnect token to the server to tell it to release resources associated with the disconnected session.

When the AFP client user no longer needs to communicate with the server, the AFP client issues an `FPLogout` command to terminate the session.

The `FPMapID` and `FPMapName` commands are used for directory access control. The `FPMapID` command obtains the user or group name corresponding to a given User or Group ID. The `FPMapName` command converts a user or group name to the corresponding User or Group ID.

The `FPChangePassword` command changes a user's password.

The `FPGetUserInfo` command retrieves information about a user.

The `FPGetSrvrMsg` command retrieves log in and server messages from the server.

Volume Commands

AFP provides the following volume-level commands:

- `FPOpenVol`
- `FPCloseVol`
- `FPGetVolParms`
- `FPSetVolParms`

- FPFlush
- FPCatSearch and FPCatSearchExt

After obtaining the volume names through the FPGetSrvrParms command, the AFP client sends an FPOpenVol command for each volume to which it wants to gain access. If a volume has a password, it must be supplied at this time. The command returns the requested volume parameters, including the volume identifier, *VolumeID*.

The volume identifier is used in all subsequent commands to identify the volume for which the commands apply and remains valid until the session is terminated by calling FPLogout or the volume is closed by calling FPVolClose.

After obtaining the volume's volume identifier, the AFP client can obtain the volume's parameters by calling FPGetVolParms. The AFP client can also change the volume's parameters by calling FPSetVolParms.

The FPFlush command requests that the server flush (write to disk) any data associated with a particular volume.

The FPCatSearch and FPCatSearchExt commands search a volume for files that match specified criteria. The FPCatSearchExt command differs from the FPCatSearch command in that FPCatSearchExt is prepared to handle the larger values that may be returned for searches on volumes greater than 4 GB in size.

Directory Commands

AFP provides these commands for working on directories:

- FPSetDirParms
- FPOpenDir
- FPCloseDir
- FPEnumerate, FPEnumerateExt, and FPEnumerateExt2
- FPCreateDir

The FPSetDirParms command allows the AFP client to modify a directory's parameters. To obtain a directory's parameters from the file server, the AFP client uses the FPGetFileDirParms command, which is described in the section "Combined Directory and File Commands" (page 74).

On variable Directory ID volumes, the AFP client uses the FPOpenDir command to open a directory on and retrieve its Directory ID. The Directory ID is used in subsequent commands to enumerate the directory or to obtain access to its offspring. For variable Directory ID volumes, the FPOpenDir command is the only way to retrieve the Directory ID. Calling FPGetFileDirParms, FPEnumerate, FPEnumerateExt, or FPEnumerateExt2 to retrieve the Directory ID on such volumes causes an error to be returned.

On a fixed Directory ID volume, calling FPGetFileDirParms, FPEnumerate, FPEnumerateExt, or FPEnumerateExt2 is the preferred way to obtain a Directory ID, although calling FPOpenDir also works.

The AFP client can close directories on variable Directory ID volumes by sending the FPCloseDir command, which invalidates the corresponding Directory ID.

The AFP client uses the FPEnumerate, FPEnumerateExt, and FPEnumerateExt2 commands to list, or enumerate, the files and directories contained within a specified directory. In reply to this command, the server returns a list of directory or file parameters corresponding to those offspring. The FPEnumerateExt command differs from the FPEnumerate command in that the FPEnumerateExt command is prepared to handle larger values that may be returned when volumes are larger than 4 GB in size. The FPEnumerateExt2 command differs from the FPEnumerate command in that the StartIndex and MaxReplySize components to the FPEnumerateExt2 command are longs, allowing you to specify larger values than can be specified by the FPEnumerate and FPEnumerateExt commands.

Directories are created by the FPCreateDir command.

File Commands

AFP provides these commands for working on files:

- FPSetFileDirParms
- FPCreateFile
- FPCopyFile
- FPCreateID
- FPDeleteID
- FPResolveID
- FPExchangeFiles

The AFP client uses the FPSetFileParms command to modify a specified file’s parameters, the FPCreateFile command to create a file, and the FPCopyFile command to copy a file that exists on a volume managed by a server to any other volume managed by that server. To obtain a specified file’s parameters, the AFP client uses the FPGetFileDirParms command, discussed in the next section.

The FPCreateID command creates a unique File ID for an existing file, and FPDeleteID removes a File ID.

The FPResolveID command uses a File ID to retrieve information about a file.

The FPExchangeFiles command preserves existing file IDs when an application performs a Save or a Save As operation.

Combined Directory and File Commands

AFP provides five commands that operate on both files and directories:

- FPGetFileDirParms
- FPSetFileDirParms
- FPRename
- FPDelete
- FPMoveAndRename

The AFP client uses the FPGetFileDirParms command to retrieve the parameters associated with a given file or directory. When it uses this command, the AFP client does not need specify whether the CNode is a file or directory; the file server indicates the CNode's type in response to this command.

The FPSetFileDirParms command is used to set the parameters of a file or directory. When the AFP client uses this command, it need not specify whether the object is a file or directory. This command allows the AFP client to set only those parameters that are common to both types of CNodes.

The FPRename command is used to rename files and directories.

The FPDelete command is used to delete a file or directory. A file can be deleted only if it is not open; a directory can be deleted only if it is empty.

The FPMoveandRename command is used to move a file or a directory from one parent directory to another on the same volume. The moved CNode can renamed at the same time.

Fork Commands

AFP provides these fork-level commands:

- FPGetForkParms
- FPSetForkParms
- FPOpenFork
- FPRead and FPReadExt
- FPWrite and FPWriteExt
- FPFlushFork
- FPByteRangeLock and FPByteRangeLockExt
- FPCloseFork

The AFP client uses the FPGetForkParms command to read a fork's parameters.

The FPSetForkParms command is used to modify a fork's parameters.

The FPOpenFork command is used to open either fork of an existing file. This command returns an open fork reference number, which is used in subsequent commands for this open fork.

The FPRead and FPReadExt commands are used to read the contents of the fork. The FPReadExt command differs from the FPRead command in that the FPReadExt command is prepared to handle large values that may be returned for volumes greater than 4 GB in size.

The FPWrite and FPWriteExt commands are used to write to a fork. The FPWriteExt command differs from the FPWrite command in that the FPWriteExt command is prepared to handle the large values that are required for writing to volumes greater than 4 GB in size.

The FPFlushFork command is used to request that server write to disk any of the fork's data that is in the server's internal buffers.

The FPByteRangeLock and FPByteRangeLockExt commands are used to lock ranges of bytes in the fork. The FPByteRangeLockExt command differs from the FPByteRangeLock command in that the FPByteRangeLockExt command is prepared to handle large values that are required for locking ranges on volumes greater than 4 GB in size. Locks allow multiple users to share a file's open fork. Locking a range of bytes prevents other AFP clients from reading or writing data within the specified range. If an AFP client locks a byte range, that range is reserved for exclusive manipulation by the client that placed the lock.

The FPCloseFork command is used to close an open fork. This command invalidates the open fork reference number that was assigned when the fork was opened.

Desktop Database Commands

An AFP client uses the following commands to read and write information stored in the server's Desktop database:

- FPOpenDT
- FPCloseDT
- FPAddIcon
- FPGetIcon
- FPGetIconInfo
- FPAddAPPL
- FPRemoveAPPL
- FPGetAPPL
- FPAddComment
- FPRemoveComment
- FPGetComment

Before any other Desktop database commands can be sent, the AFP client must send an FPOpenDT command. This command returns a reference number to be used in all subsequent commands on the Desktop database.

When access to the Desktop database is no longer needed, the AFP client makes an FPCloseDT command.

FPAddIcon adds a new icon to the Desktop database, and FPGetIcon retrieves the bitmap for a given icon as specified by its file creator and type. FPGetIconInfo retrieves a description of an icon. This command can be used to determine the set of icons associated with a given application. Successive FPGetIconInfo commands return information on all icons associated with a given file creator.

FPAddAPPL adds an APPL mapping for the specified application and its file creator. FPRemoveAPPL removes the specified application from the list of APPL mappings corresponding to its file creator. It is the AFP client's responsibility to add and remove APPL mappings for applications that are added to or removed from the volume, respectively. For applications that are moved or renamed, the AFP client should remove the old APPL mapping before the operation and add a new APPL mapping with the updated information after the operation has been completed successfully.

FPGetAPPL returns the next APPL mapping in the Desktop database's list of applications that correspond to a given file creator.

FPAddComment stores a comment string associated with a particular file or directory on the volume. When adding a comment for a file or directory that already has an associated comment, the existing comment is replaced.

FPRemoveComment removes the comment associated with a particular file or directory. FPGetComment retrieves the comment associated with a particular file or directory.

Apple Filing Protocol Reference

Framework:

The Apple Filing Protocol (AFP) API describes the request blocks that an AFP client sends to an AFP server and the reply blocks that an AFP server sends to an AFP client in response to a request block.

Note that all values exchanged between an AFP client and an AFP server are sent over the network in network byte order.

Apple Filing Protocol Commands

FPAddAPPL

Adds an APPL mapping to the Desktop database.

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

Parameter Descriptions

`CommandCode`

`kFPAddAPPL` (53).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`DirectoryID`

Ancestor Directory ID.

`FileCreator`

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

`ApplTag`

User-defined tag stored with the APPL mapping.

`PathType`

Type of name in pathName, where 1 indicates short names, 2 indicates long names, and 3 indicates Unicode names.

`Pathname`

Pathname to the application corresponding to the APPL mapping being added; a string if the `PathType` is 1 or 2 or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-1 (page 81) for other possible result codes.

`Reply block`

None.

Discussion

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

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

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

The user must have previously called `FPOpenDT` (page 204) for the corresponding volume. In addition, the application must be present in the specified directory before this command is sent.

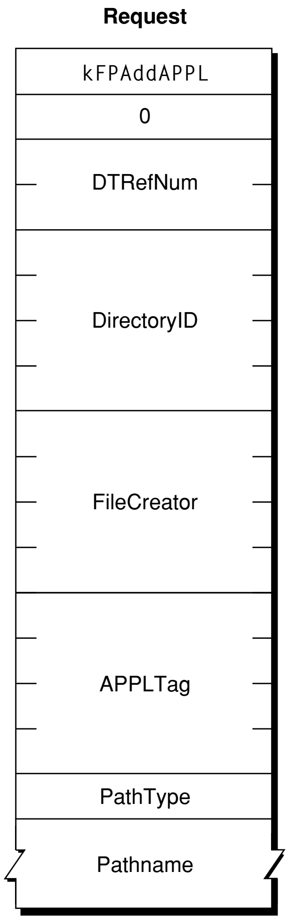
Table 3-1 (page 81) lists the result codes for the `FPAddAPPL` command.

Table 3-1 Result codes for the `FPAddAPPL` command

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

Figure 3-1 shows the request block for the `FPAddAPPL` command.

Figure 3-1 Request block for `FPAddAPPL`



FPAddComment

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

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

Parameter Descriptions

`CommandCode`

`kFPAddComment` (56).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of name in `Pathname`, where 1 indicates short names, 2 indicates long names, and 3 indicates Unicode names.

`Pathname`

Pathname to the file or directory with which the comment is to be associated; a string if `PathType` is 1 or 2 or an AFPName if `PathType` is 3.

`Comment`

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

`Result`

`kFPNoErr` if no error occurred. See Table 3-2 (page 84) for other possible result codes.

`ReplyBlock`

None.

Discussion

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

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

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

The user must have previously called `FPOpenDT` (page 204) for the corresponding volume. In addition, the specified file or directory must be present in the specified directory before this command is sent.

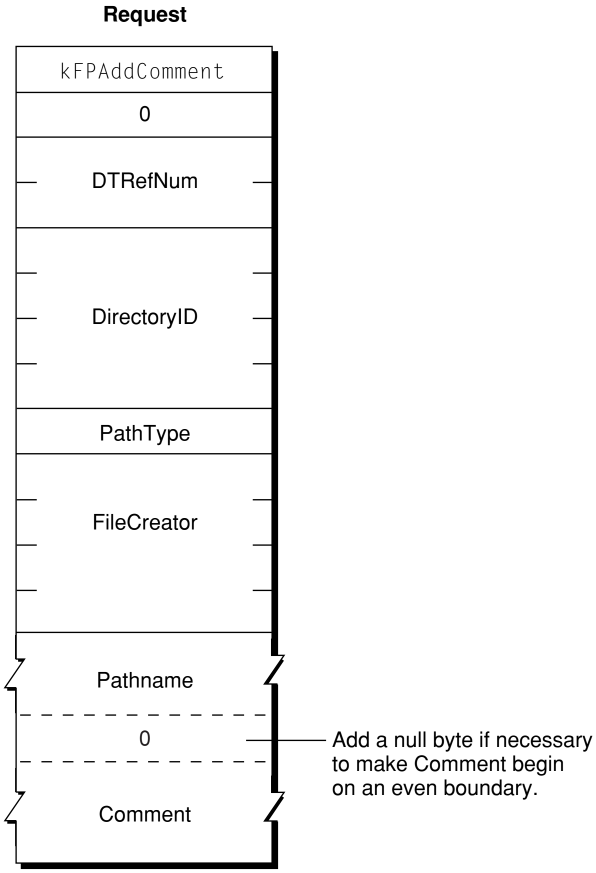
Table 3-2 (page 84) lists the result codes for the `FPAddComment` command.

Table 3-2 Result codes for the `FPAddComment` command

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

Figure 3-2 (page 84) shows the request block for the `FPAddComment` command.

Figure 3-2 Request block for `FPAddComment`



FPAddIcon

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

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

Parameter Descriptions

`CommandCode`

`kFPAddIcon` (192).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`FileCreator`

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

`FileType`

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

`IconType`

Type of icon that is to be added.

`Pad`

Pad byte.

`IconTag`

Tag information to be stored with the icon.

`BitmapSize`

Size of the bitmap for this icon.

`Result`

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

`ReplyBlock`

None.

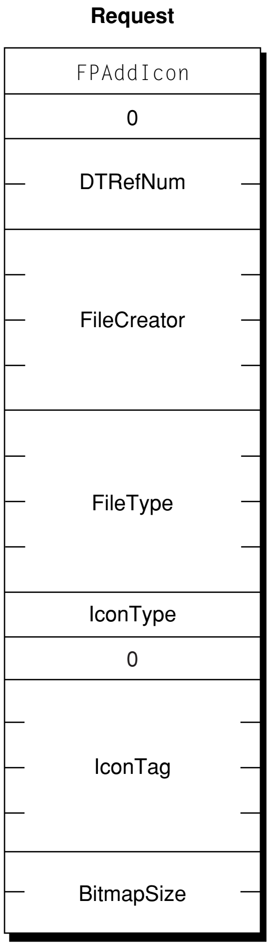
Discussion

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

The user must have previously called FPOpenDT (page 204) for the corresponding volume.

Figure 3-3 (page 87) shows the request block for the FPAddIcon command.

Figure 3-3 Request block for the FPAddIcon command



FPByteRangeLock

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

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

Parameter Descriptions

`CommandCode`

kFPByteRangeLock (1).

`Pad`

Pad byte.

`DTRefNum`

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

`OForkRefNum`

Open fork reference number.

`Offset`

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

`Length`

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

`Result`

kFPNoErr if no error occurred. See Table 3-3 (page 89) for possible result codes.

`ReplyBlock`

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

Discussion

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

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

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

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

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

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

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

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

Table 3-3 (page 89) lists the result codes for the `FPByteRangeLock` command.

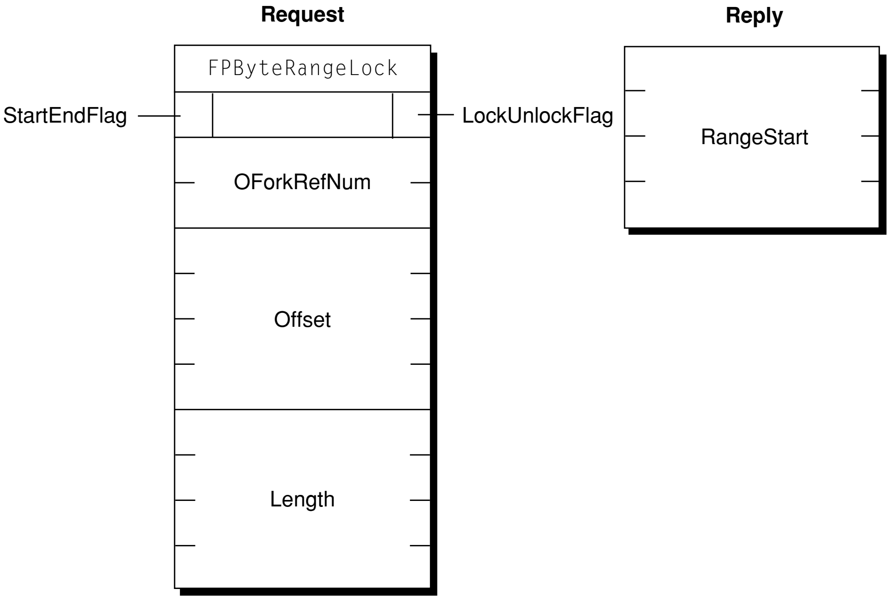
Table 3-3 Result codes for the `FPByteRangeLock` command

Result code	Explanation
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPNoMoreLocks`	Server’s maximum lock count has been reached.

Result code	Explanation
kFPParamErr	Session reference number or open fork reference number is unknown; a combination of the StartEndFlag bit and Offset specifies a range that starts before byte zero.
kFPRangeNotLocked	User tried to unlock a range that is locked by another user or that is not locked at all.
kFPRangeOverlap	User tried to lock some or all of a range that the user has already locked.

Figure 3-4 (page 90) shows the request and reply blocks for the FPByteRangeLock command.

Figure 3-4 Request and reply blocks for the FPByteRangeLock command



FPByteRangeLockExt

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

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

Parameter Descriptions

CommandCode

kFPByteRangeLockExt (59).

`Pad`

Pad byte.

`DTRefNum`

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

`OForkRefNum`

Open fork reference number.

`Offset`

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

`Length`

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

`Result`

`kFPNoErr` if no error occurred. See Table 3-4 (page 92) for possible result codes.

`ReplyBlock`

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

Discussion

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

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

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

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

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

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

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

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

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

Table 3-4 (page 92) lists the result codes for the FPByteRangeLockExt command.

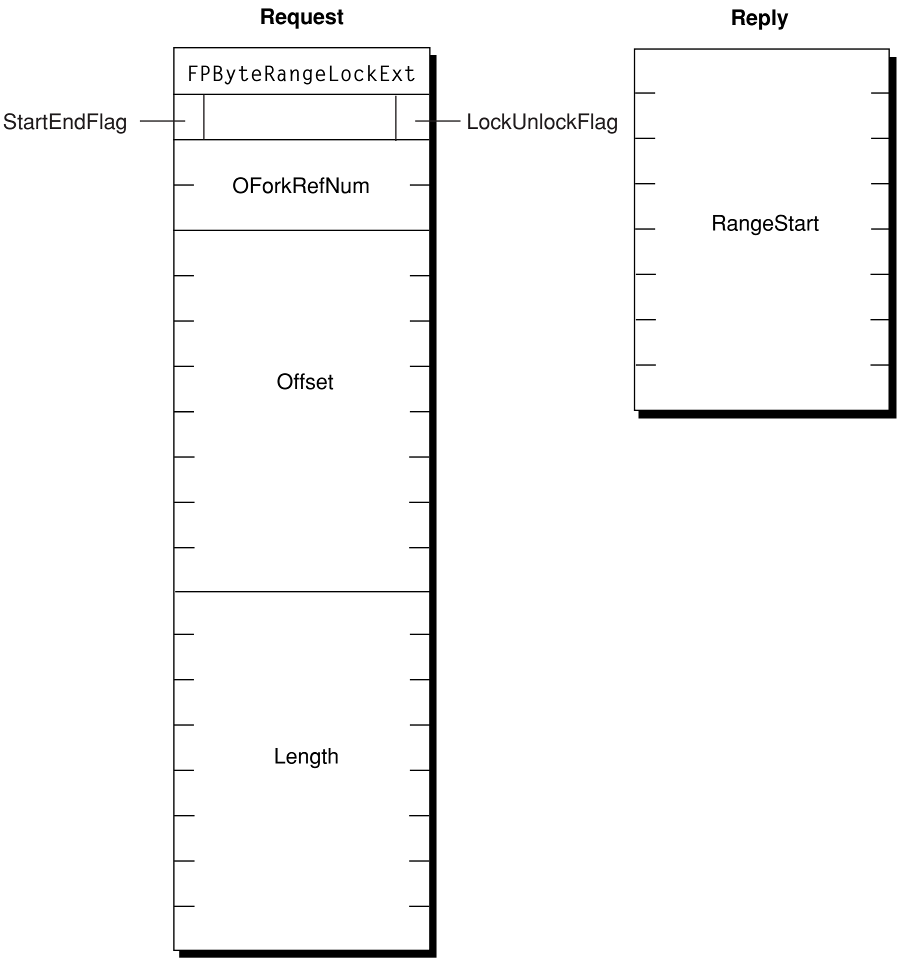
Table 3-4 Result codes for the FPByteRangeLockExt command

Result code	Explanation
kFPLockErr	Some or all of the requested range is locked by another user.

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

Figure 3-5 (page 93) shows the request and reply blocks for the `FPByteRangeLockExt` command.

Figure 3-5 Request and reply blocks for the `FPByteRangeLockExt` command



FPCatSearch

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

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

Parameter Descriptions

`CommandCode`

kFPCatSearch (43).

`Pad`

Pad byte.

`VolumeID`

The ID of the volume to search.

`Reserved`

Reserved; must be zero.

`ReqMatches`

The maximum number of matches to return.

`CatalogPosition`

Current position in the catalog.

`FileRsltBitmap`

File bitmap describing the file parameters to get or null to get directory parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the FPGetFileDirParms (page 156) command with some restrictions described in the Discussion section. For bit definitions for this bitmap, see "File Bitmap" (page 253).

`DirectoryRsltBitmap`

Directory bitmap describing the directory parameters to get or null to get file parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the DirectoryBitmap parameter of the FPGetFileDirParms (page 156) command with some restrictions described in the Discussion section. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

`ReqBitmap`

Directory and file parameters that are to be searched. For directory parameters only, see Figure 3-6 (page 97). For file parameters only, see Figure 3-7 (page 97). For directory and file parameters, see Figure 3-8 (page 98).

`Specification1`

Search criteria lower bounds and values.

`Specification2`

Optional search criteria upper bounds and masks.

`Length`

Length of this request block.

`Result`

`kFPNoErr` if no error occurred. See Table 3-5 (page 98) for possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-6 (page 99) for the format of the reply block.

Discussion

This command searches a volume for files and directories that match the specified criteria and returns an array of records that describe the matches that were found. The criteria can include most parameters in the File bitmap, the Directory bitmap, or both bitmaps, that are defined for the FPGetFileDirParms (page 156) command. Parameters for the matching files and directories are returned. These parameters can also be any of those specified by the FPGetFileDirParms command.

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

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

The fields in Specification1 and Specification2 have different uses:

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

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

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

The ReqBitmap parameter specifies the directory and file parameters to be searched. The low-order word of ReqBitmap is the same as low-order word of the File bitmap and the Directory bitmap used by the FPGetFileDirParms (page 156) command, with the exception of the Short Name parameter, which cannot be searched. The high bit of the high-order word of ReqBitmap indicates whether the search should match on full names or partial names (0 = full name, 1 = partial name). There is no equivalent to the fsSBNegate bit used by the Macintosh File Manager's `PBCatSearch` function.

Figure 3-6 (page 97) shows parameters this command can search when it is searching directories only.

Figure 3-6 Parameters `FPCatSearch` searches when searching directories only

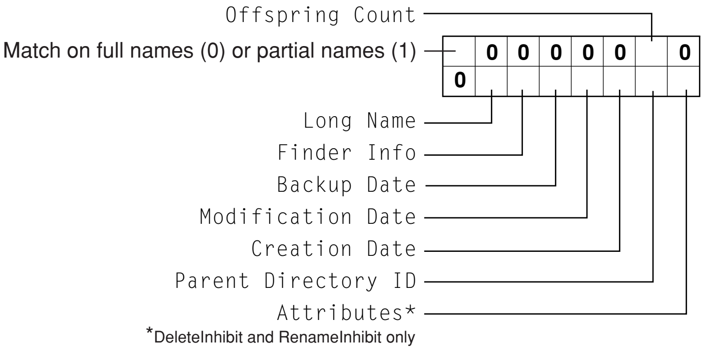


Figure 3-7 (page 97) shows parameters this command can search when it is searching files only.

Figure 3-7 Parameters `FPCatSearch` searches when searching files only

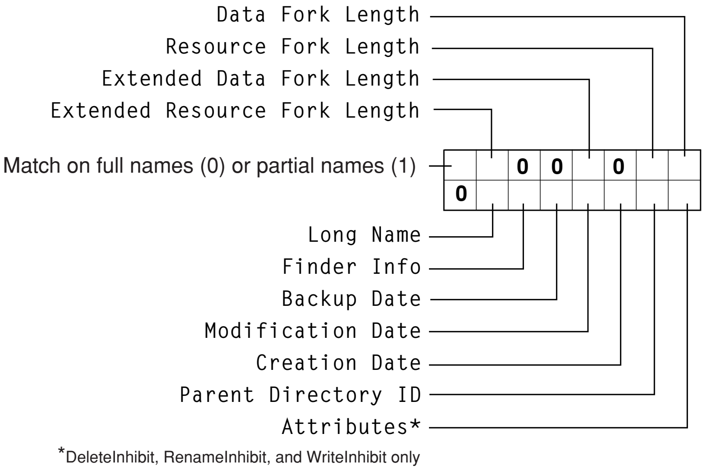
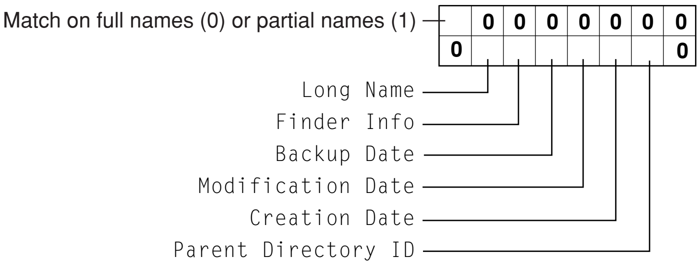

Data Fork Length
Resource Fork Length
Extended Data Fork Length
Extended Resource Fork Length
Match on full names (0) or partial names (1)
0
0
0
0
Long Name
Finder Info
Backup Date
Modification Date
Creation Date
Parent Directory ID
Attributes*
*DeleteInhibit, RenameInhibit, and WriteInhibit only

Figure 3-8 (page 98) shows parameters this command can search when it is searching both directories and files.

Figure 3-8 Parameters FPCatSearch searches when searching directories and files



Before sending this command, the user must call FPOpenVol (page 209) for the volume that is to be searched.

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

Table 3-5 lists the result codes for the FPCatSearch command.

Table 3-5 Result codes for the FPCatSearch command

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

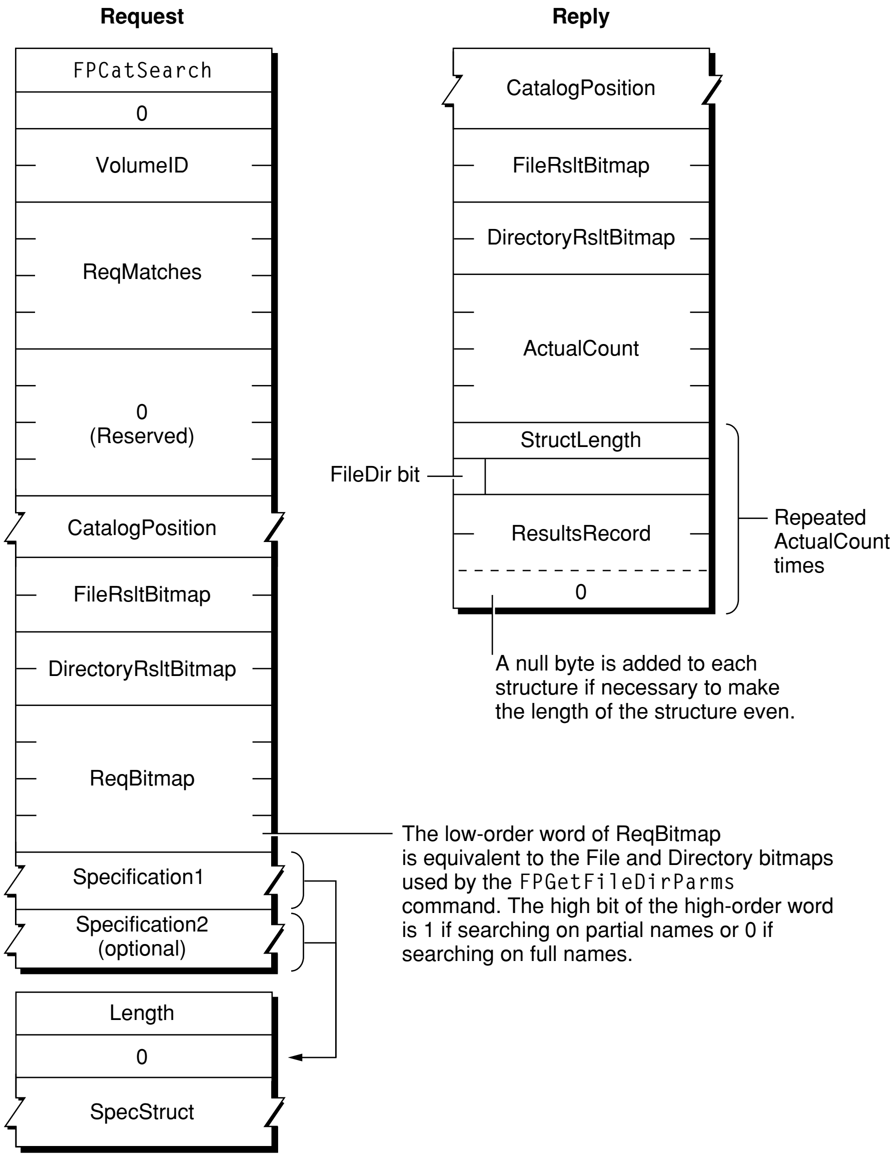
Table 3-6 describes the reply block for the FPCatSearch command.

Table 3-6 Reply block for the FPCatSearch command

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

Figure 3-9 shows the request and reply blocks for the FPCatSearch command.

Figure 3-9 Request and reply blocks for the `FPCatSearch` command


Request
FPCatSearch
0
VolumeID
ReqMatches
0
(Reserved)
CatalogPosition
FileRsltBitmap
DirectoryRsltBitmap
ReqBitmap
Specification1
Specification2
(optional)
Length
0
SpecStruct
Reply
CatalogPosition
FileRsltBitmap
DirectoryRsltBitmap
ActualCount
StructLength
FileDir bit
ResultsRecord
0
Repeated
ActualCount
times
A null byte is added to each structure if necessary to make the length of the structure even.
The low-order word of ReqBitmap is equivalent to the File and Directory bitmaps used by the FPGetFileDirParms command. The high bit of the high-order word is 1 if searching on partial names or 0 if searching on full names.

FPCatSearchExt

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

```
byte CommandCode
byte Pad
short VolumeID
long ReqMatches
long Reserved
16 bytes CatalogPosition
short FileRsltBitmap
short DirectoryRsltBitmap
long ReqBitmap
```

```
Specification1
Specification2
unsigned char Length
```

Parameter Descriptions

`CommandCode`

`kFPCatSearchExt` (67).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`ReqMatches`

Maximum number of matches to return.

`Reserved`

Reserved; must be zero.

`CatalogPosition`

Current position in the catalog.

`FileRsltBitmap`

Bitmap describing the file parameters to get or null to get directory parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 156) command with some restrictions described later in this section. For bit definitions for this bitmap, see "File Bitmap" (page 253).

`DirectoryRsltBitmap`

Bitmap describing the directory parameters to get or null to get file parameters only. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 156) command with some restrictions described later in this section. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

`ReqBitmap`

Directory and file parameters that are to be searched. For directory parameters, see Figure 3-10 (page 104). For file parameters, see Figure 3-11 (page 104). For directory and file parameters, see Figure 3-12 (page 105).

`Specification1`

Search criteria lower bounds and values.

`Specification2`

Optional search criteria upper bounds and masks.

`Length`

Length of this request block.

`Result`

`kFPNoErr` if no error occurred. See Table 3-7 (page 105) for possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-8 (page 106) for the format of the reply block.

Discussion

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

This command differs from the FPCatSearch (page 94) command in that FPCatSearchExt is prepared to handle longer search results that can occur when searching volumes that are more than 4 GB in size.

The criteria can include most parameters in the File bitmap, the Directory bitmap, or both bitmaps, that are defined for the FPGetFileDirParms (page 156) command. Parameters for the matching files and directories are returned. These parameters can also be any of those specified by the FPGetFileDirParms command.

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

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

The fields in Specification1 and Specification2 have different uses:

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

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

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

The ReqBitmap parameter specifies the directory and file parameters to be searched. The low-order word of ReqBitmap is the same as low-order word of the File bitmap and the Directory bitmap in FPGetFileDirParms (page 156), with the exception of the Short Name parameter, which cannot be searched. The high bit of the high-order word of ReqBitmap indicates whether the search should match on full names or partial names (0 = full name, 1 = partial name). There is no equivalent to the fsSBNegate bit used by the Macintosh File Manager's PBCatSearch function.

Figure 3-10 (page 104) shows parameters this command can search when it is searching directories only.

Figure 3-10 Parameters FPCatSearchExt searches when searching directories only

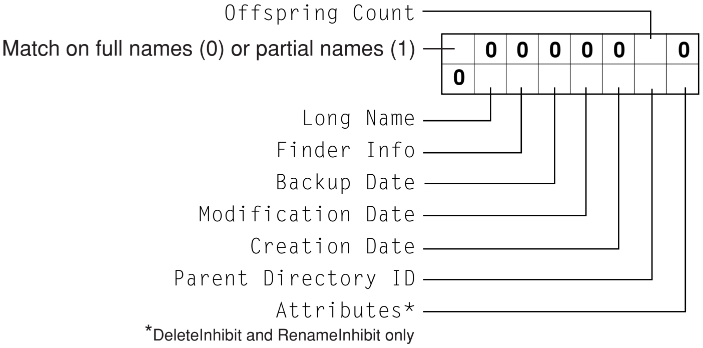


Figure 3-7 (page 97) shows parameters this command can search when it is searching files only.

Figure 3-11 Parameters FPCatSearchExt searches when searching files only

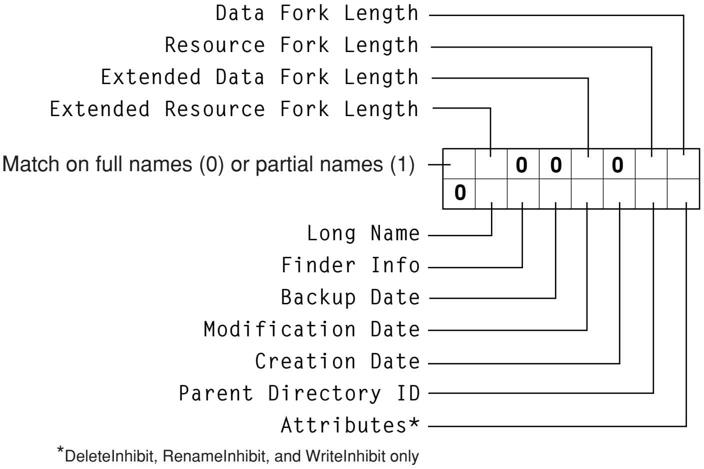
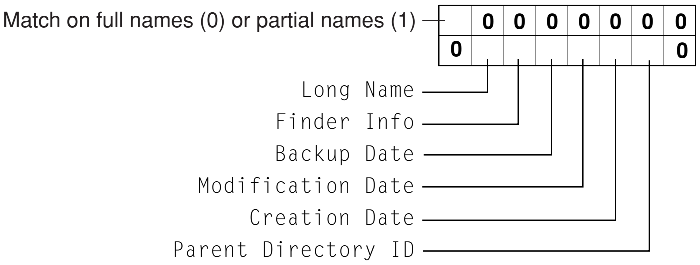

Data Fork Length
Resource Fork Length
Extended Data Fork Length
Extended Resource Fork Length
Match on full names (0) or partial names (1)
0
0
0
0
Long Name
Finder Info
Backup Date
Modification Date
Creation Date
Parent Directory ID
Attributes*
*DeleteInhibit, RenameInhibit, and WriteInhibit only

Figure 3-8 (page 98) shows parameters this command can search when it is searching both directories and files.

Figure 3-12 Parameters `FPCatSearchExt` searches when searching directories and files



Before sending this command, the user must call `FPOpenVol` (page 209) for the volume that is to be searched.

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

Table 3-7 (page 105) lists the result codes for the `FPCatSearchExt` command.

Table 3-7 Result codes for the `FPCatSearchExt` command

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

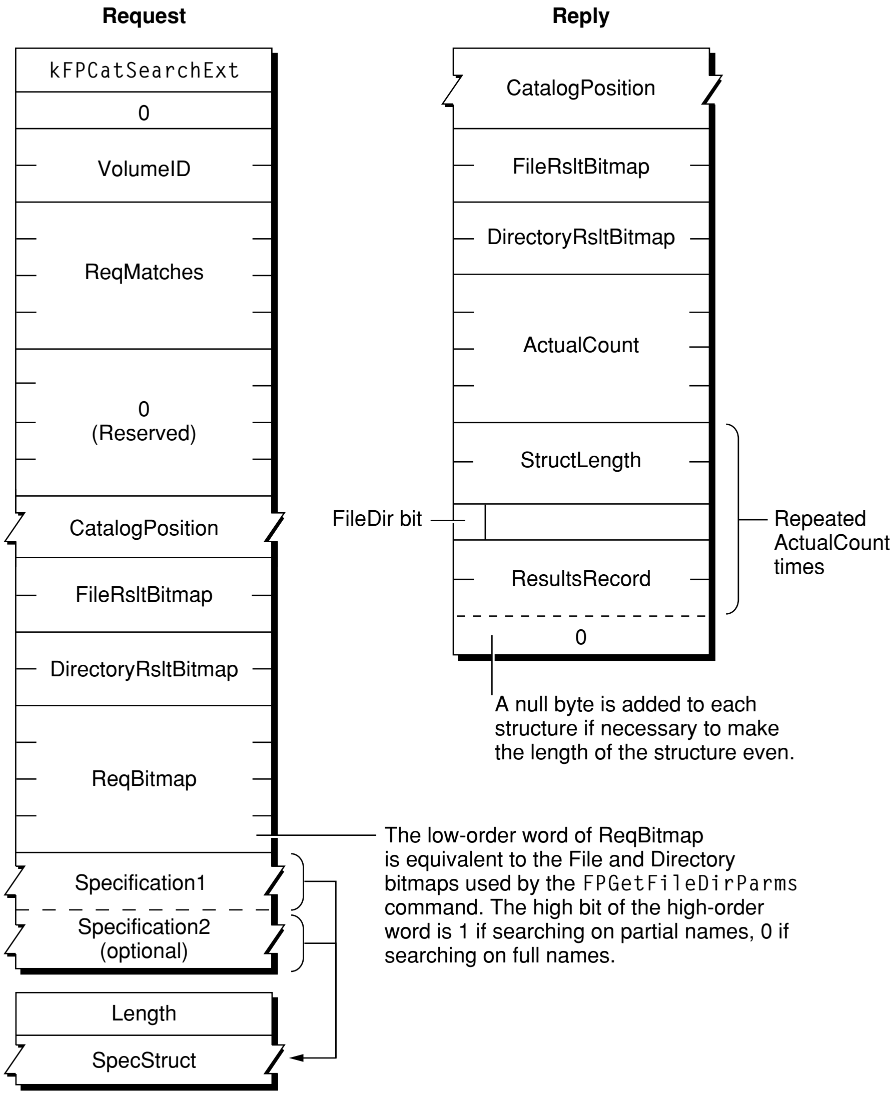
Table 3-8 describes the reply block for the `FPCatSearchExt` command.

Table 3-8 Reply block for the FPCatSearchExt command

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

Figure 3-13 shows the request and reply blocks for the FPCatSearchExt command.

Figure 3-13 Request and reply blocks for the FPCatSearchExt command



A null byte is added to each structure if necessary to make the length of the structure even.

The low-order word of ReqBitmap is equivalent to the File and Directory bitmaps used by the FPGetFileDirParms command. The high bit of the high-order word is 1 if searching on partial names, 0 if searching on full names.

FPChangePassword

Allows users to change their passwords.

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

Parameter Descriptions

CommandCode

kFPChangePassword (36).

`Pad`

Pad byte.

`UAM`

String specifying the UAM to uses.

`UserName`

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

`UserAuthInfo`

UAM-specific information.

`Result`

`kFPNoErr` if no error occurred. See Table 3-9 (page 109) for other possible result codes.

`ReplyBlock`

None.

Discussion

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

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

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

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

If the UAM is DHCAST128, the AFP client must call FPChangePassword twice. The first time, the AFP client calls FPChangePassword to send the user name and a random number that has been encrypted. The server replies with an ID, a random number, and a nonce/server signature value encrypted by a session key. The AFP client calls FPChangePassword again, this time sending the user name and the ID returned by the server. The client also sends the nonce incremented by one, the new password, and the old password, all encrypted by the session key. For information on using the DHX UAM to change passwords, see the section "DHX and Changing a Password" in the "Introduction" section.

Servers are not required to support this command. Call FPGetSrvrInfo (page 169) to determine whether a server supports this command.

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

Table 3-9 lists the result codes for the FPChangePassword command.

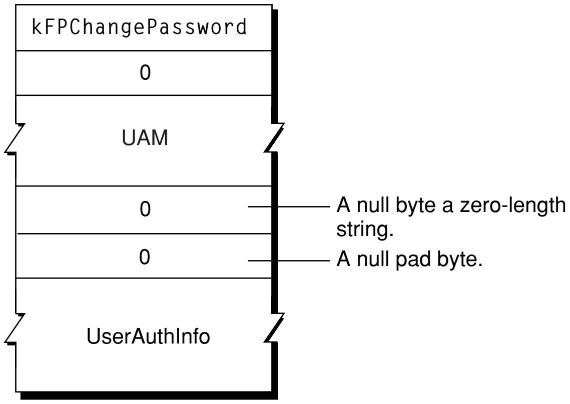
Table 3-9 Result codes for the FPChangePassword command

Result code	Explanation
kFPNoErr	No error occurred.
kFPCallNotSupported	Server does not support this command.
kFPUserNotAuth	UAM failed (the specified old password doesn't match) or no user is logged in yet for the specified session.
kFPBadUAM	Specified UAM is not a UAM that FPChangePassword supports.

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.
kFPParamErr	User name is null, exceeds the UAM's user name length limit, or does not exist.
kFPPwdSameErr	User attempted to change his or her password to the same password that he or she previously had. This error occurs only if the password expiration feature is enabled on the server.
kFPPwdTooShortErr	User password is shorter than the server's minimum password length, or user attempted to change password to a password that is shorter than the server's minimum password length.
kFPPwdPolicyErr	New password does not conform to the server's password policy.
kFPMiscErr	Non-AFP error occurred.

Figure 3-14 (page 110) shows the request block for the FPChangePassword command.

Figure 3-14 Request block for the FPChangePassword command



FPCloseDir

Closes a directory and invalidates its Directory ID.

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

Parameter Descriptions

CommandCode

kFPCloseDir (3).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Directory ID.

Result

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

ReplyBlock

None.

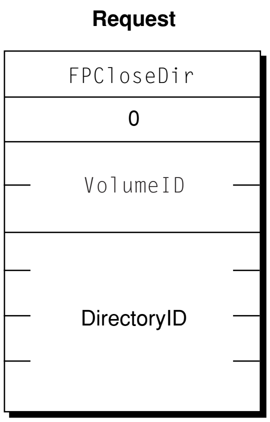
Discussion

This command invalidates the Directory ID specified by DirectoryID.

This command should be used only for variable Directory ID volumes. The user must have previously called FPOpenVol (page 209) for this volume and FPOpenDir (page 201) for this directory.

Figure 3-15 (page 111) shows the request block for the FPCloseDir command.

Figure 3-15 Request block for the FPCloseDir command



FPCloseDT

Closes a volume's Desktop database.

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

Parameter Descriptions

`CommandCode`

`kFPCloseDT` (49).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`Result`

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

`ReplyBlock`

None.

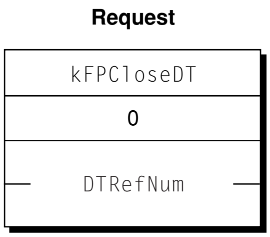
Discussion

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

The user must first have sent a successful `FPOpenDT` (page 204) command.

Figure 3-16 (page 112) shows the request block for the `FPCloseDT` command.

Figure 3-16 Request block for the `FPCloseDT` command



FPCloseFork

Closes a fork.

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

Parameter Descriptions

`CommandCode`

`kFPCloseFork` (4).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Result

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

ReplyBlock

None.

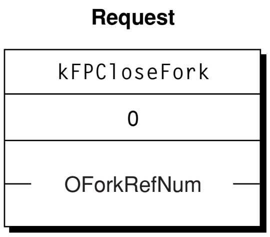
Discussion

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

The user must first have sent a successful FPOpenFork (page 205) command.

Figure 3-17 (page 113) shows the request block for the FPCloseFork command.

Figure 3-17 Request block for the FPCloseFork command



FPCloseVol

Closes a volume.

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

Parameter Descriptions

CommandCode

kFPCloseVol (2).

Pad

Pad byte.

VolumeID

Volume ID.

Result

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

ReplyBlock

None.

Discussion

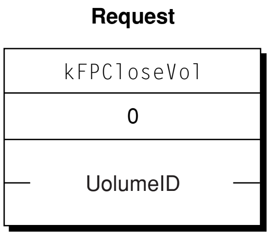
This command invalidates the specified Volume ID but does not necessarily close all open files on a volume before closing the volume, so you should close all open files before calling FPCloseVol.

The user must first have sent a successful FPOpenVol (page 209) command for this volume.

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

Figure 3-18 (page 114) shows the request block for the FPCloseVol command.

Figure 3-18 Request block for the FPCloseVol command



FPCopyFile

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

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

Parameter Descriptions

CommandCode

kFPCopyFile (5).

Pad

Pad byte.

SourceVolumeID

Source Volume ID.

SourceDirectoryID

Source ancestor Directory ID.

DestVolumeID

Destination Volume ID.

DestDirectoryID

Destination ancestor Directory ID.

SourcePathType

Type of names in SourcePathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

SourcePathname

Pathname of the file to be copied (cannot be null). SourcePathname is a string if SourcePathType is 1 or 2, or an AFPName if PathType is 3.

DestPathType

Type of names in DestPathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

DestPathname

Pathname to the destination parent directory (may be null). DestPathname is a string if DestPathType is 1 or 2, or an AFPName if DestPathType is 3.

NewType

Type of name in NewName, where 1 indicates Short Name, 2 indicates Long Name, and 3 indicates Unicode name.

NewName

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

Result

kFPNoErr if no error occurred. See Table 3-10 (page 116) for other possible result codes.

ReplyBlock

None.

Discussion

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

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

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

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

The user must first have sent a successful FPOpenVol (page 209) command for both the source volume and the destination volume.

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

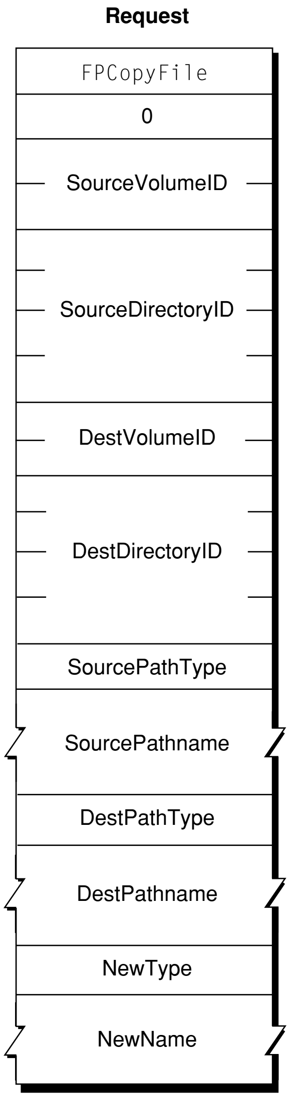
Table 3-10 lists the result codes for the FPCopyFile command.

Table 3-10 Result codes for the FPCopyFile command

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

Figure 3-19 (page 117) shows the request block for the FPCopyFile command.

Figure 3-19 Request block for the `FPCopyFile` command



FPCreateDir

Creates a new directory.

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

Parameter Descriptions

`CommandCode`

`kFPCreateDir` (6).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname, including the name of the new directory (cannot be null). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-11 (page 119) for other possible result codes.

`ReplyBlock`

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

Discussion

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

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

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

The user must have previously called `FPOpenVol` (page 209) for this volume.

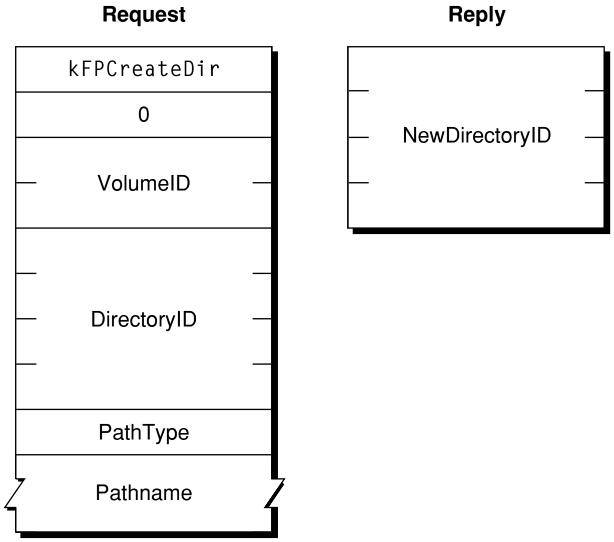
Table 3-11 lists the result codes for the `FPCreateDir` command.

Table 3-11 Result codes for the `FPCreateDir` command

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

Figure 3-20 (page 119) shows the request and reply blocks for the `FPCreateDir` command.

Figure 3-20 Request and reply blocks for the `FPCreateDir` command



FPCreateFile

Creates a new file.

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

Parameter Descriptions

`CommandCode`

`kFPCreateFile` (7).

`Flag`

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

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname, including the name of the new file (cannot be null). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

Result

kFPNoErr if no error occurred. See Table 3-12 (page 122) for other possible result codes.

ReplyBlock

None.

Discussion

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

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

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

The user must have previously called FPOpenVol (page 209) for this volume.

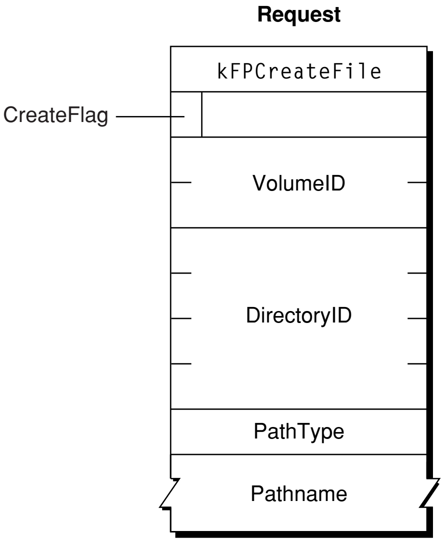
Table 3-12 lists the result codes for the FPCreateFile command.

Table 3-12 Result codes for the FPCreateFile command

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

Figure 3-21 (page 122) shows the request block for the FPCreateFile command.

Figure 3-21 Request block for the FPCreateFile command



FPCreateID

Creates a unique File ID for a file.

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

Parameter Descriptions

`CommandCode`

`kFPCreateID` (39).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

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

`PathType`

Type of names in Pathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Name of the file that is the target of the File ID (that is, the filename of the file for which a File ID is being created). Pathname is a string if PathType is 1 or 2, or an AFPName if PathType is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-13 (page 125) for other possible result codes.

`ReplyBlock`

None.

Discussion

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

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

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

The user must have previously called `FPOpenVol` (page 209) for this volume.

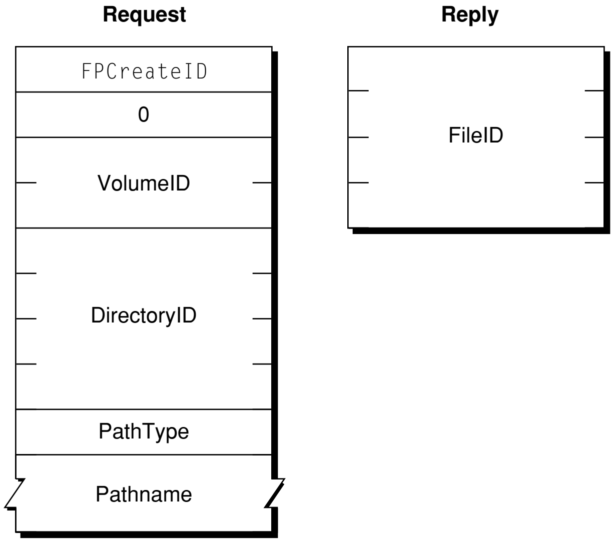
Table 3-13 lists the result codes for the `FPCreateID` command.

Table 3-13 Result codes for the `FPCreateID` command

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

Figure 3-22 (page 125) shows the request block for the `FPCreateID` command.

Figure 3-22 Request block for the `FPCreateID` command



FPDelete

Deletes a file or directory.

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

Parameter Descriptions

`CommandCode`

`kFPDelete` (8).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname of the file or directory to be deleted (may be null if a directory is to be deleted). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-14 (page 127) for other possible result codes.

`ReplyBlock`

None.

Discussion

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

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

The user must have previously called `FPOpenVol` (page 209) for this volume.

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

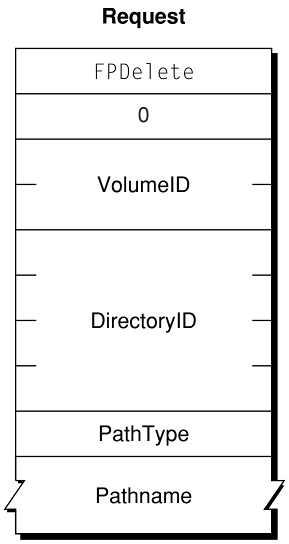
Table 3-14 lists the result codes for the `FPDelete` command.

Table 3-14 Result codes for the `FPDelete` command

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

Figure 3-23 (page 127) shows the request block for the `FPDelete` command.

Figure 3-23 Request block for the `FPDelete` command



FPDeleteID

Invalidates all instances of the specified File ID.

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

Parameter Descriptions

CommandCode

kFPDeleteID (40).

Pad

Pad byte.

VolumeID

Volume ID.

FileID

File ID that is to be deleted.

Result

kFPNoErr if no error occurred. See Table 3-15 for other possible result codes.

ReplyBlock

None.

Discussion

This command deletes the specified File ID, which was created by an earlier call to FPCreateID (page 123).

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

The user must have previously called FPOpenVol (page 209) for this volume.

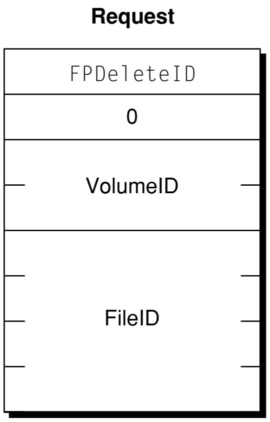
Table 3-15 lists the result codes for the FPDeleteID command.

Table 3-15 Result codes for the FPDeleteID command

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

Figure 3-24 (page 129) shows the request block for the FPDeleteID command.

Figure 3-24 Request block for the FPDeleteID command



FPDisconnectOldSession

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

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

Parameter Descriptions

`CommandCode`

`kFPDisconnectOldSession` (65).

`Pad`

Pad byte.

`Type`

Volume ID.

`TokenLength`

Length of `Token`.

`Token`

Token previous obtained by calling `FPGetSessionToken` (page 166).

`Result`

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

`ReplyBlock`

None.

Discussion

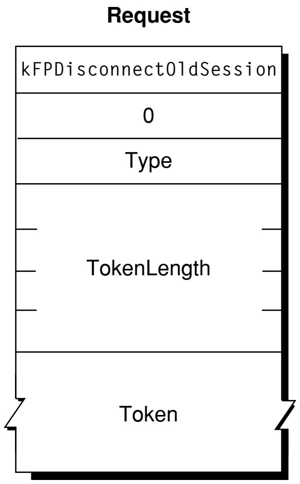
This command disconnects the session identified by the Token parameter, which was obtained by previously calling `FPGetSessionToken` (page 166) and transfers the resources of the old session to the new session.

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

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

Figure 3-25 (page 131) shows the request block for the `FPDisconnectOldSession` command.

Figure 3-25 Request block for the `FPDisconnectOldSession` command



FPEnumerate

Lists the contents of a directory.

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

Parameter Descriptions

`CommandCode`

`kFPEnumerate` (9).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Identifier for the directory to list.

`FileBitmap`

Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 156) command and can be null. For bit definitions for this bitmap, see "File Bitmap" (page 253).

`DirectoryBitmap`

Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 156) command and can be null. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

`ReqCount`

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

`StartIndex`

Directory offspring index.

`MaxReplySize`

Maximum size of the reply block.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the desired directory. `Pathname` is a string if PathType is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-16 (page 133) for other possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-17 (page 134) for the format of the reply block.

Discussion

This command enumerates a directory as specified by the input parameters. This command differs from the FPEnumerateExt (page 135) and FPEnumerateExt2 (page 139) commands in that it is not able to handle values that may be returned when volumes are larger than 4 GB in size.

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

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

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

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

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

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

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

The user must have previously called FPOpenVol (page 209) for this volume.

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

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

Table 3-16 lists the result codes for the FPEnumerate command.

Table 3-16 Result codes for the FPEnumerate command

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

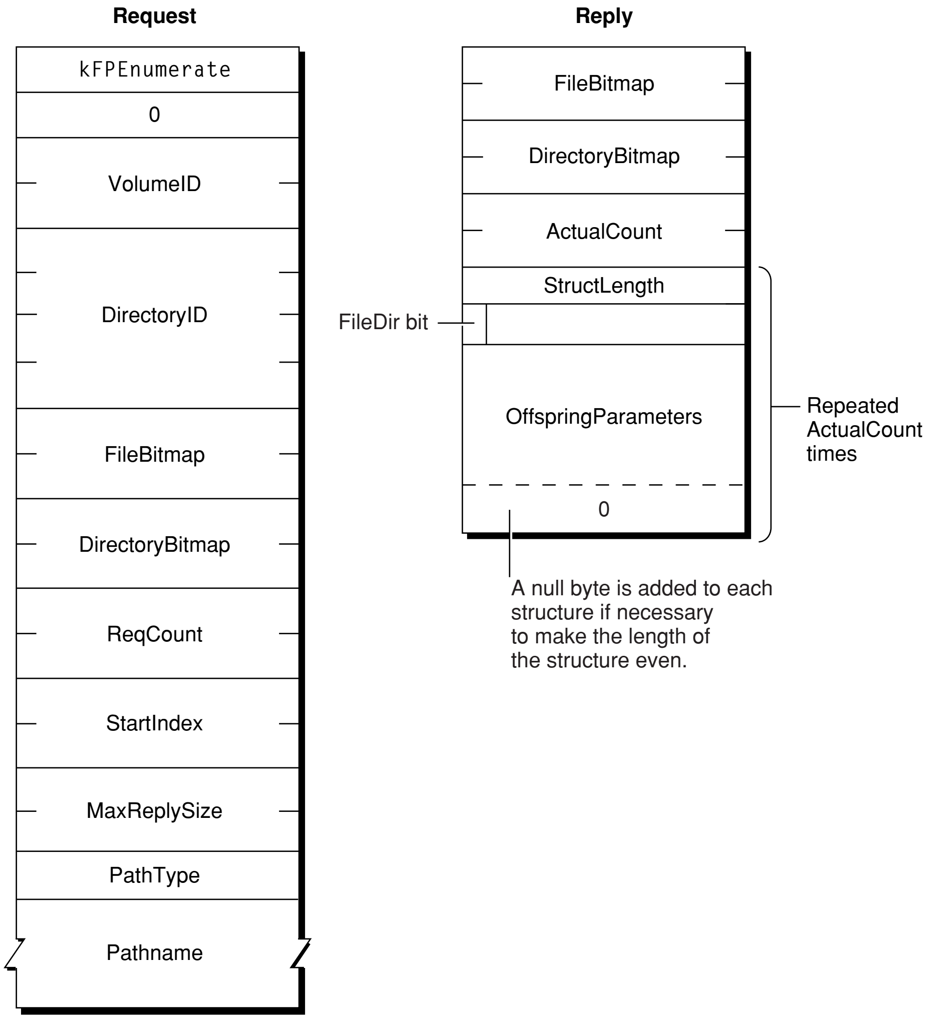
Table 3-17 describes the reply block for the FPEnumerate command.

Table 3-17 Reply block for the FPEnumerate command

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

Figure 3-26 shows the request and reply blocks for the FPEnumerate command.

Figure 3-26 Request and reply blocks for the `FPEnumerate` command



FPEnumerateExt

Lists the contents of a directory.

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

Parameter Descriptions

`CommandCode`

`kFPEnumerateExt` (66).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Identifier for the directory to list.

`FileBitmap`

Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 156) command and can be null. For bit definitions for this bitmap, see "File Bitmap" (page 253).

`DirectoryBitmap`

Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 156) command and can be null. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

`ReqCount`

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

`StartIndex`

Directory offspring index.

`MaxReplySize`

Maximum size of the reply block.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the desired directory. `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-18 (page 138) for other possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-19 (page 138) for the format of the reply block.

Discussion

This command enumerates a directory as specified by the input parameters. This command differs from the FPEnumerate (page 131) command in that this command is prepared to handle values that may be returned when volumes are larger than 4 GB in size. This command also differs from the FPEnumerateExt2 (page 139) command in that StartIndex and MaxReplySize are shorts (instead of longs), which may limit the number of entries in a single directory that can be listed. The reply block for this command is the same as the reply block for FPEnumerateExt2.

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

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

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

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

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

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

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

The user must have previously called FPOpenVol (page 209) for this volume.

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

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

Table 3-18 lists the result codes for the FPEnumerateExt command.

Table 3-18 Result codes for the FPEnumerateExt command

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

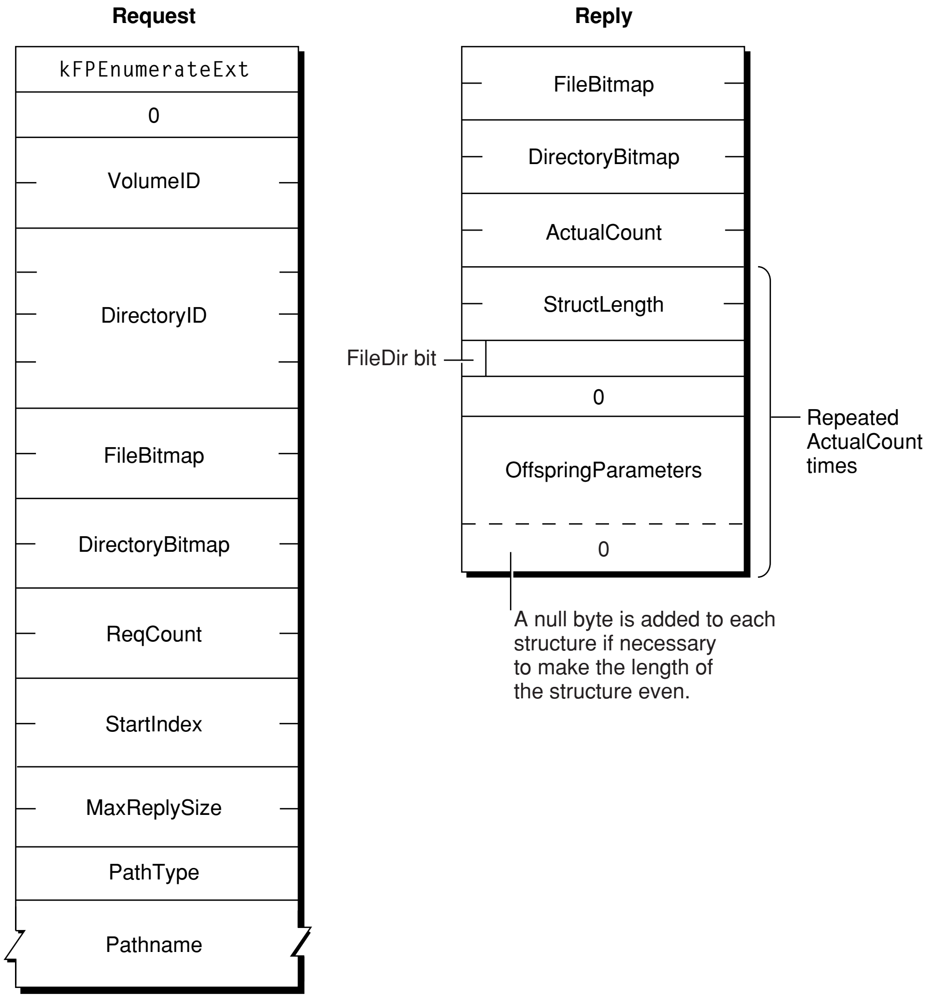
Table 3-19 describes the reply block for the FPEnumerateExt command.

Table 3-19 Reply block for the FPEnumerateExt command

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

Figure 3-27 shows the request and reply blocks for the FPEnumerateExt command.

Figure 3-27 Request and reply blocks for the `FPEnumerateExt` command



FPEnumerateExt2

Lists the contents of a directory.

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

Parameter Descriptions

CommandCode

kFPEnumerateExt2 (68).

Pad

Pad byte.

VolumeID

Volume ID.

DirectoryID

Identifier for the directory to list.

FileBitmap

Bitmap describing the parameters to return if the enumerated offspring is a file. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the FPGetFileDirParms (page 156) command and can be null. For bit definitions for this bitmap, see "File Bitmap" (page 253).

DirectoryBitmap

Bitmap describing the parameters to return if the enumerated offspring is a directory. Set the bit that corresponds to each desired parameter. This bitmap is the same as the DirectoryBitmap parameter of the FPGetFileDirParms (page 156) command and can be null. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

ReqCount

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

StartIndex

Directory offspring index.

StartIndex

Maximum size of the reply block.

PathType

Type of names in Pathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

Pathname

Pathname to the desired directory. Pathname is a string if PathType is 1 or 2, or an AFPName if PathType is 3.

Result

kFPNoErr if no error occurred. See Table 3-20 (page 141) for other possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-21 (page 142) for the format of the reply block.

Discussion

This command enumerates a directory as specified by the input parameters. This command differs from the FPEnumerateExt (page 135) command in that StartIndex and MaxReplySize are longs instead of shorts, thereby allowing this command to list more entries in a single directory. The reply block for this command is the same as the reply block for the FPEnumerateExt command.

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

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

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

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

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

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

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

The user must have previously called FPOpenVol (page 209) for this volume.

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

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

Table 3-20 lists the result codes for the FPEnumerateExt2 command.

Table 3-20 Result codes for the FPEnumerateExt2 command

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

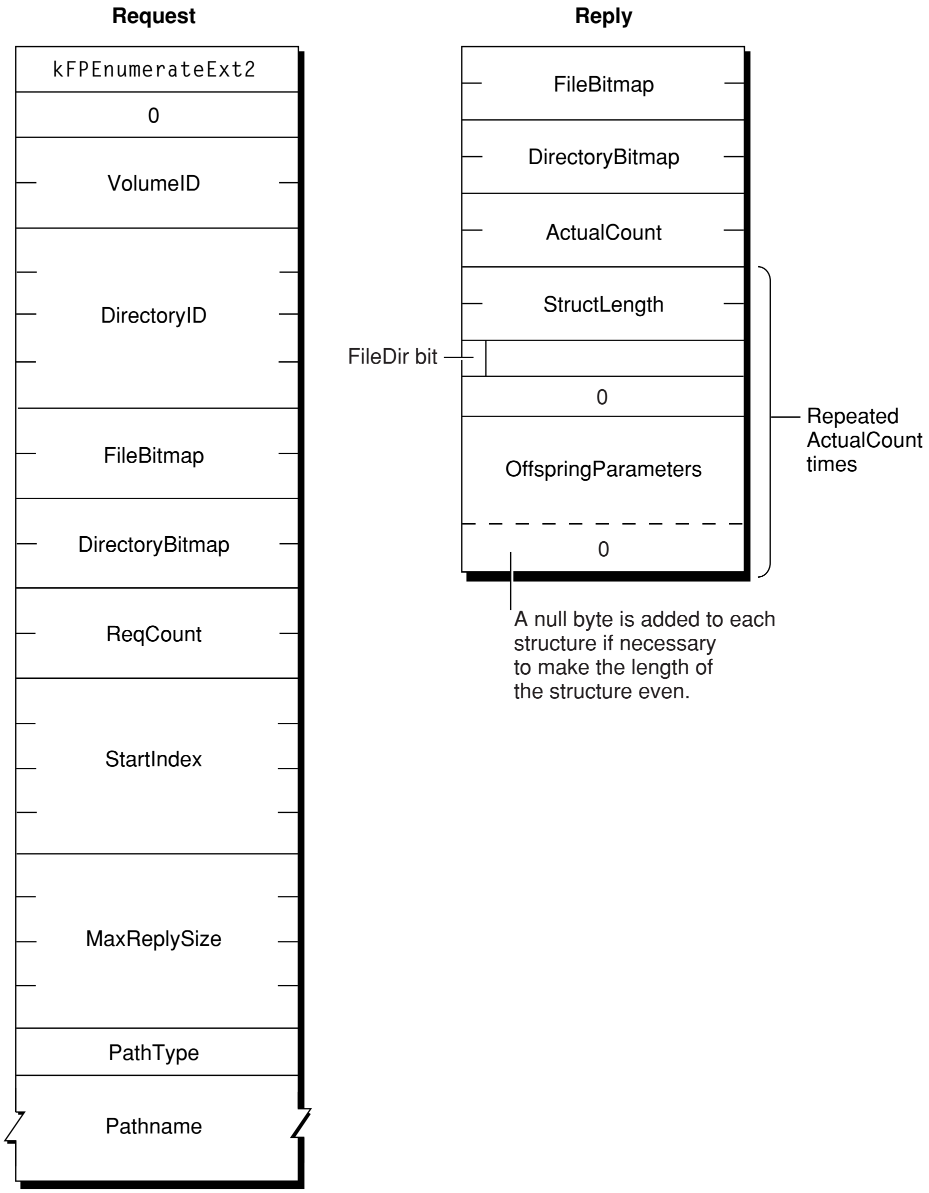
Table 3-21 describes the reply block for the FPEnumerateExt2 command.

Table 3-21 Reply block for the FPEnumerateExt2 command

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

Figure 3-28 shows the request and reply blocks for the FPEnumerateExt2 command.

Figure 3-28 Request and reply blocks for the `FPEnumerateExt2` command



FPExchangeFiles

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

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

```
byte DestPathType
string DestPathname
```

Parameter Descriptions

`CommandCode`

`kFPExchangeFiles` (42).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`SourceDirectoryID`

Identifier of the directory containing the source file.

`DestDirectoryID`

Identifier of the directory containing the destination file.

`SourcePathType`

Type of names in `SourcePathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`SourcePathname`

Pathname of the source file, which is a string if `SourcePathType` is 1 or 2, or an AFPName if `SourcePathType` is 3.

`DestPathType`

Type of names in `DestPathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`DestPathname`

Pathname of the source file, which is a string if `SourcePathType` is 1 or 2, or an AFPName if `SourcePathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-22 (page 145) for other possible result codes.

`ReplyBlock`

None.

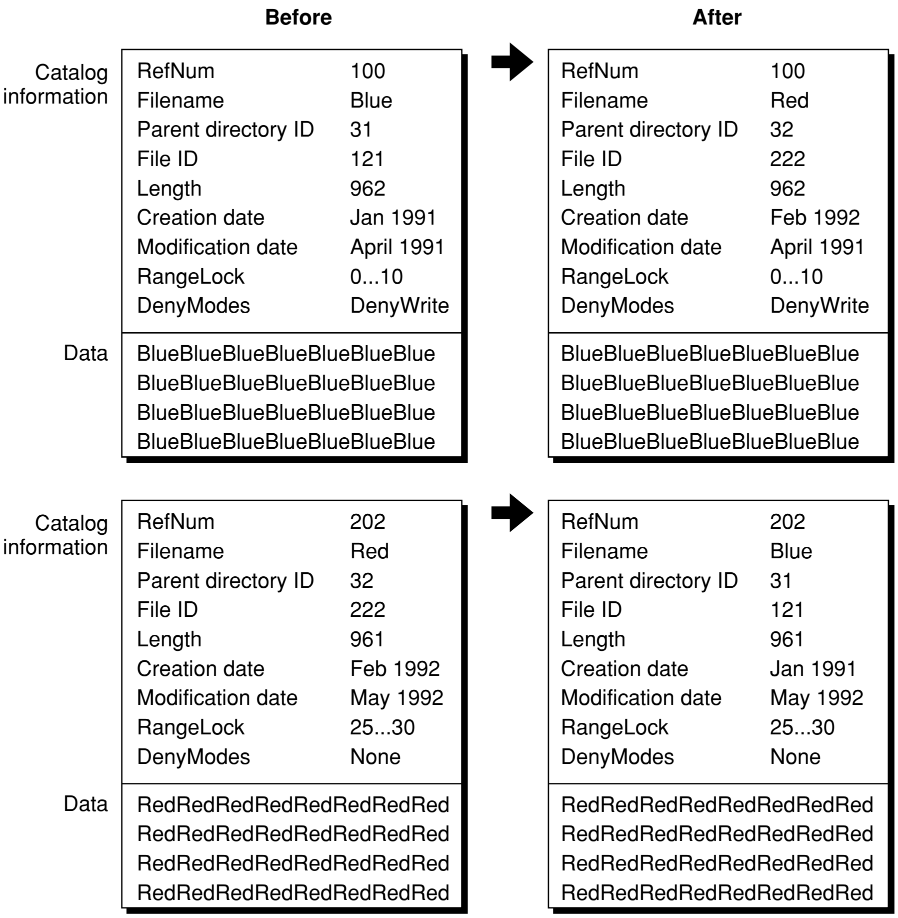
Discussion

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

Before calling this command, you must call `FPOpenVol` (page 209) for the volume on which the files reside.

Figure 3-29 (page 145) shows the results of an exchange operation between two files named Blue and Red.

Figure 3-29 Example of exchanging files



	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	
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.

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

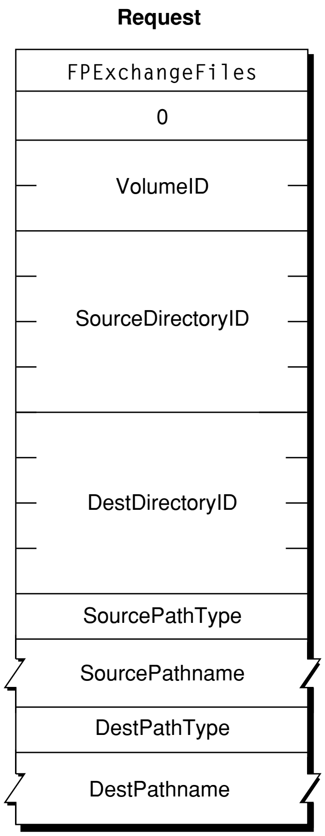
Table 3-22 lists the result codes for the `FPExchangeFiles` command.

Table 3-22 Result codes for the `FPExchangeFiles` command

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

Figure 3-30 shows the request block for the FPExchangeFiles command.

Figure 3-30 Request block for the FPExchangeFiles command



FPFlush

Writes any volume data that has been modified.

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

Parameter Descriptions

CommandCode

kFPFlush (10).

Pad

Pad byte.

VolumeID

Volume ID.

Result

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

ReplyBlock

None.

Discussion

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

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

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

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

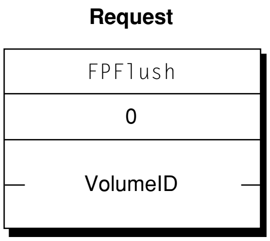
Notice that only the filename, Parent Directory ID, File ID, and creation dates are exchanged. Byte-range locks and deny modes still apply to the same file reference number and data.

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

The user must have previously called `FPOpenVol` (page 209) for this volume.

Figure 3-31 shows the request block for the `FPFlush` command.

Figure 3-31 Request block for the `FPFlush` command



FPFlushFork

Writes any data buffered from previous write commands.

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

Parameter Descriptions

`CommandCode`

`kFPFlushFork` (11).

`Pad`

Pad byte.

OForkRefNum

Open fork reference number.

Result

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

ReplyBlock

None.

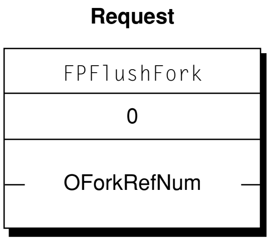
Discussion

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

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

Figure 3-32 shows the request block for the FPFlushFork command.

Figure 3-32 Request block for the FPFlushFork command



FPGetAPPL

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

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

Parameter Descriptions

CommandCode

kFPGetAPPL (55).

Pad

Pad byte.

DTRefNum

Desktop database reference number.

`FileCreator`

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

`APPLIndex`

Index of the APPL mapping to be retrieved.

`Result`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-23 (page 151) for the format of the reply block.

Discussion

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

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

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

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

The user must have previously called FPOpenDT (page 204) for the corresponding volume.

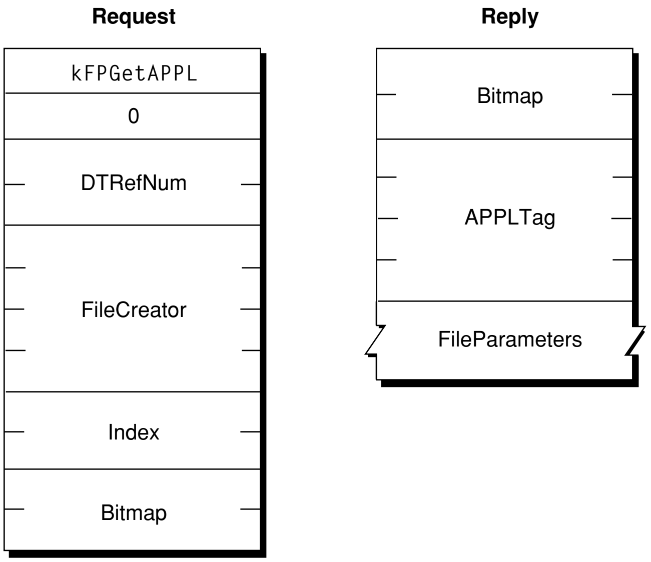
Table 3-23 describes the reply block for the FPGetAPPL command.

Table 3-23 Reply block for the FPGetAPPL command

Name and size	Data
Bitmap (short)	Bitmap describing the parameters of the application file to return. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the FPGetFileDirParms (page 156) command. For bit definitions for the File bitmap, see Table 1-7 in "File Bitmap" (page 253).
APPLTag (long)	Tag information associated with the APPL mapping.
FileParameters	Requested file parameters.

Figure 3-33 shows the request and reply blocks for the FPGetAPPL command.

Figure 3-33 Request and reply blocks for the FPGetAPPL command



FPGetAuthMethods

Gets the UAMs that an Open Directory domain supports.

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

Parameter Descriptions

`CommandCode`

`kFPGetAuthMethods` (62).

`Pad`

Pad byte.

`Flags`

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

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname of the Open Directory domain for which UAMs are to be obtained. `Pathname` is a string if `PathType` is 1 or 2, or an AFPName PathType is 3.

`Result`

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

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-24 (page 153) for the format of the reply block.

Discussion

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

The user must have previously called FPOpenDT (page 204) for the corresponding volume.

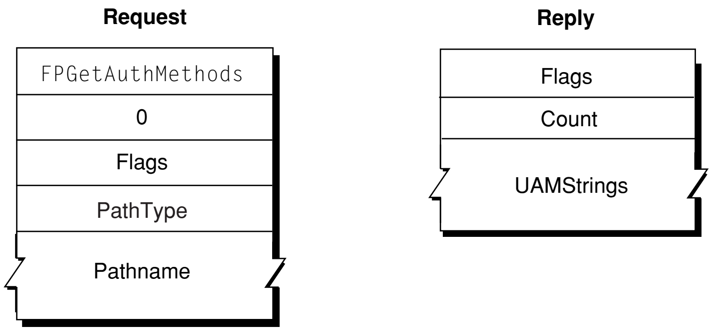
Table 3-24 describes the reply block for the FPGetAuthMethods command.

Table 3-24 Reply block for the FPGetAuthMethods command

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

Figure 3-34 shows the request and reply blocks for the FPGetAuthMethods command.

Figure 3-34 Request and reply blocks for the FPGetAuthMethods command



FPGetComment

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

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

Parameter Descriptions

`CommandCode`

`kFPGetComment` (58).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`DirectoryID`

Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to desired file or directory. `Pathname` is a string if PathType is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-25 for other possible result codes.

`ReplyBlock`

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

Discussion

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

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

The user must have previously called FPOpenDT (page 204) for the corresponding volume. In addition, the file or directory must exist before this command is sent.

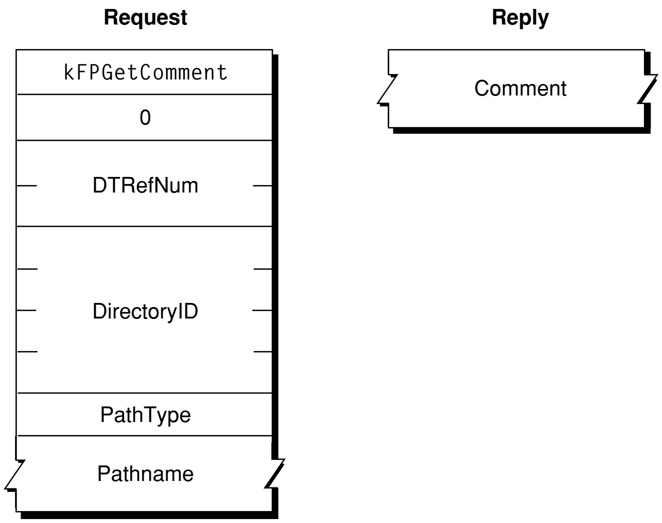
Table 3-25 lists the result codes for the FPGetComment command.

Table 3-25 Result codes for the FPGetComment command

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

Figure 3-35 shows the request and reply blocks for the FPGetComment command.

Figure 3-35 Request and reply blocks for the FPGetComment command



FPGetFileDirParms

Gets the parameters for a file or a directory.

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

Parameter Descriptions

`CommandCode`

`kFPGetFileDirParms` (34).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Directory ID.

`FileBitmap`

Bitmap describing the parameters to return for a file. Set the bit that corresponds to each desired parameter. For the bit definitions of this bitmap, see "File Bitmap" (page 253).

`DirectoryBitmap`

Bitmap describing the parameters to return for a directory. Set the bit that corresponds to each desired parameter. For the bit definitions of this bitmap, see "Directory Bitmap" (page 251).

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to desired file or directory. `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-26 (page 159) for other possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-27 (page 159) for the format of the reply block.

Discussion

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

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

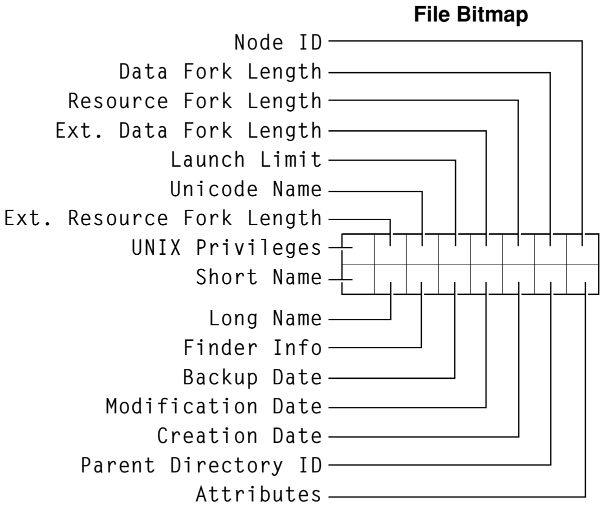
If the CNode exists and both bitmaps are null, no error is returned; FileBitmap, DirectoryBitmap, and the byte containing the FileDir bit are returned with no other parameters.

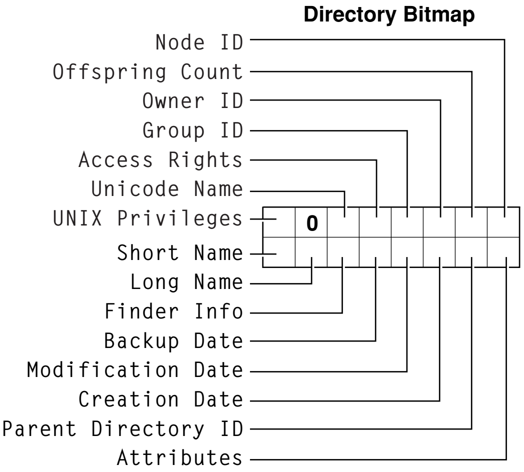
If a directory's access rights are requested, the server returns the Access Rights parameter (a four-byte quantity) containing the read, write, and search access privileges corresponding to owner, group, and everyone as well as the User Access Rights Summary byte, which indicates the privileges the current user of the AFP client has to this directory. For bit definitions of the Access Rights parameter, see "Access Rights Bitmap" (page 250).

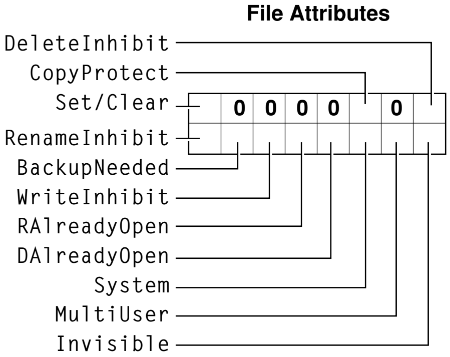
If the Offspring Count bit of the DirectoryBitmap parameter is set, the server will adjust the Offspring Count to reflect the access rights the user has to that directory. For example, if a particular directory contains three file and two directory offspring, the server will return its Offspring Count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

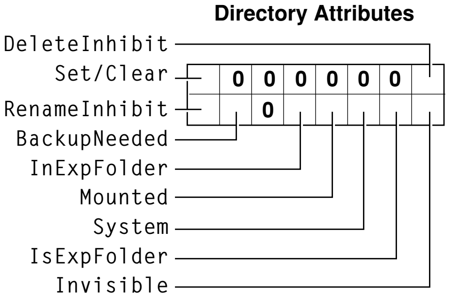
Figure 3-36 (page 157) shows the File and Directory bitmaps, the File and Directory Attributes parameters, and the Access Rights for directories.

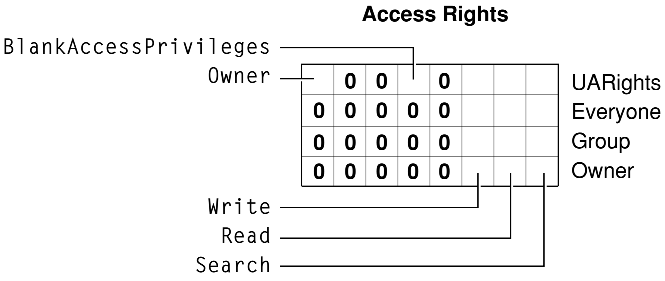
Figure 3-36 Bitmaps, Attributes, and Access Rights returned by FPGetFileDirParms











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

The user must have previously called `FPOpenVol` (page 209) for this volume.

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

Table 3-26 lists the result codes for the `FPGetFileDirParms` command.

Table 3-26 Result codes for the FPGetFileDirParms command

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

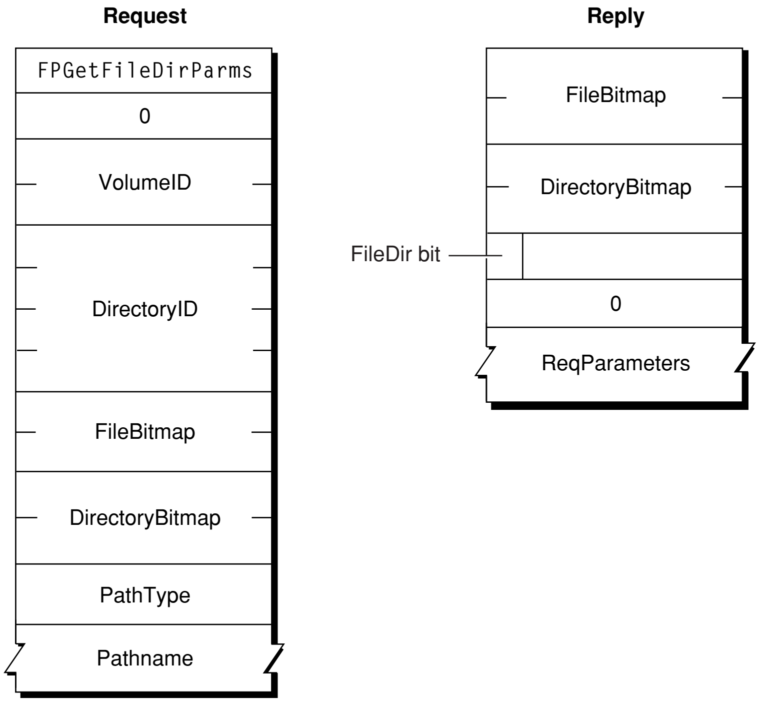
Table 3-27 describes the reply block for the FPGetFileDirParms command.

Table 3-27 Reply block for the FPGetFileDirParms command

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

Figure 3-37 shows the request and reply blocks for the FPGetFileDirParms command.

Figure 3-37 Request and reply blocks for the `FPGetFileDirParms` command



FPGetForkParms

Gets the parameters for a fork.

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

Parameter Descriptions

`CommandCode`

`kFPGetForkParms` (14).

`Pad`

Pad byte.

`OForkRefNum`

Open fork reference number.

`Bitmap`

Bitmap describing the parameters to be returned. Set the bits that correspond to each desired parameter. This bitmap is the same as the `FileBitmap` parameter of the `FPGetFileDirParms` (page 156) command. For bit definitions for this bitmap, see "File Bitmap" (page 253).

Result

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

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-28 (page 161) for the format of the reply block.

Discussion

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

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

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

The user must have previously called FPOpenFork (page 205) for this volume.

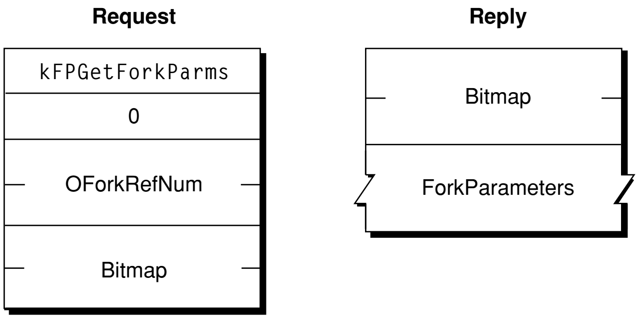
Table 3-28 describes the reply block for the FPGetForkParms command.

Table 3-28 Reply block for the FPGetForkParms command

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

Figure 3-38 shows the request and reply blocks for the FPGetForkParms command.

Figure 3-38 Request and reply blocks for the FPGetForkParms command



FPGetIcon

Gets an icon from the Desktop database.

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

Parameter Descriptions

`CommandCode`

`kFPGetIcon` (51).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`FileCreator`

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

`FileType`

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

`IconType`

Preferred icon type.

`Pad`

Pad byte.

`Length`

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

`Result`

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

`ReplyBlock`

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

Discussion

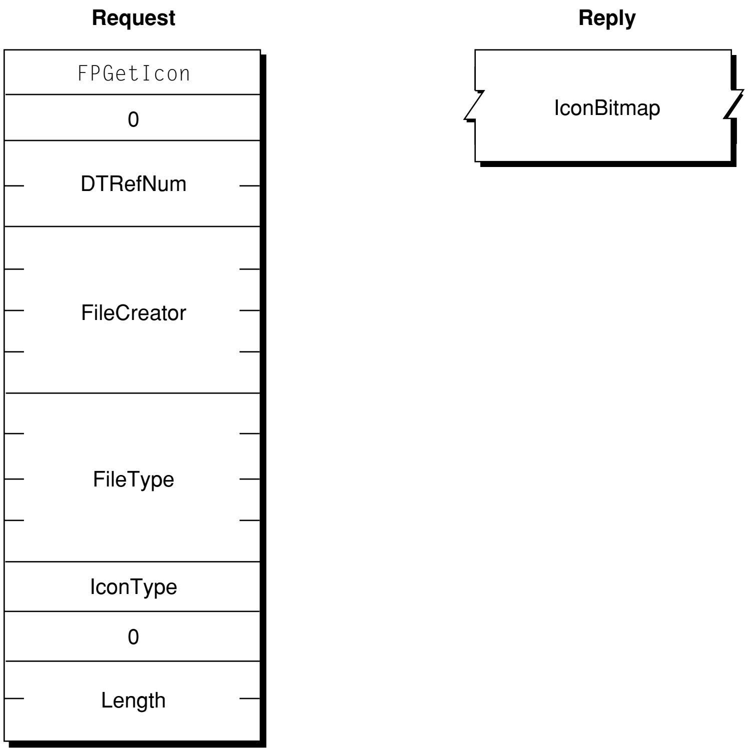
The server retrieves an icon bitmap from the Desktop database as specified by the FileCreator, FileType, and IconType parameters.

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

The user must have previously called FPOpenDT (page 204) for the corresponding volume.

Figure 3-39 shows the request and reply blocks for the FPGetIcon command.

Figure 3-39 Request and reply blocks for the FPGetIcon command



FPGetIconInfo

Gets icon information from the Desktop database.

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

Parameter Descriptions

CommandCode

kFPGetIconInfo (51).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`FileCreator`

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

`IconIndex`

Index of the requested icon.

`Result`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-29 (page 165) for the format of the reply block.

Discussion

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

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

The user must have previously called FPOpenDT (page 204) (page 204) for the corresponding volume.

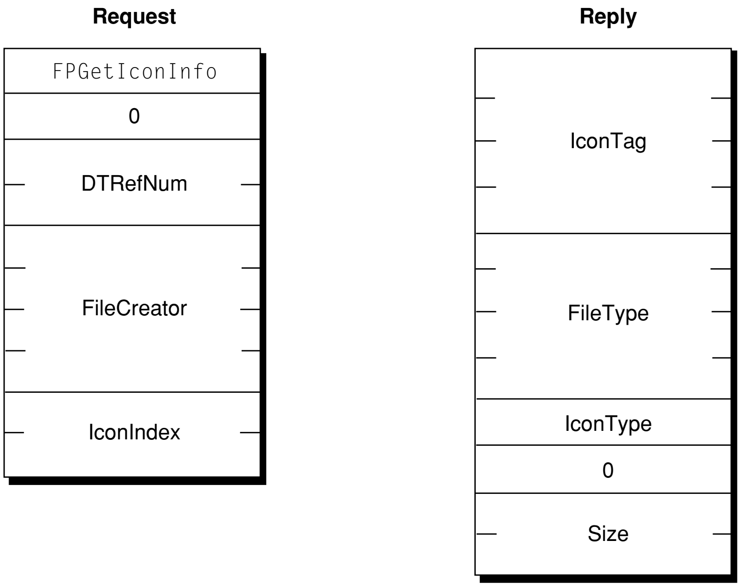
Table 3-29 describes the reply block for the FPGetIconInfo command.

Table 3-29 Reply block for the FPGetIconInfo command

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

Figure 3-40 shows the request and reply blocks for the FPGetIconInfo command.

Figure 3-40 Request and reply blocks for the FPGetIconInfo command



FPGetSessionToken

Gets a session token.

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

Parameter Descriptions

`CommandCode`

`kFPGetSessionToken` (64).

`Pad`

Pad byte.

`Type`

The value of this parameter is `kLoginWithoutID` (0) if the client supports an earlier version of AFP that does not send an `IDLength` and an `ID` parameter. It is `kLoginWithTimeAndID` (3) if the client is sending an `IDLength`, an `ID`, and a `Timestamp` parameter and the client wants its old session to be discarded. It is `kReconnWithTimeAndID` (4) if the client has just finished a successful reconnect, is sending an `IDLength`, an `ID`, and a `Timestamp` parameter, and wants the session to be updated with the `ID` parameter. It is `kGetKerberosSessionKey` (8) if the client is logging in using Kerberos v5. See "FPGetSessionToken Types" (page 260) for the enumeration that defines the constants for this parameter.

`IDLength`

Length of the `ID` parameter.

timeStamp

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

ID

A client-defined value that uniquely identifies this session.

Result

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

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-30 (page 168) for the format of the reply block.

Discussion

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

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

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

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

This command returns a session token that, with AFP 3.0 and later, is used to reconnect. If the local system is disconnected and the AFP client logs in again using the same log in information as before, the AFP client can call `FPDisconnectOldSession` (page 130), passing the session token obtained by calling `FPGetSessionToken`, to tell the server to transfer the old session to the new session.

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

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

Table 3-30 describes the reply block for the `FPGetSessionToken` command.

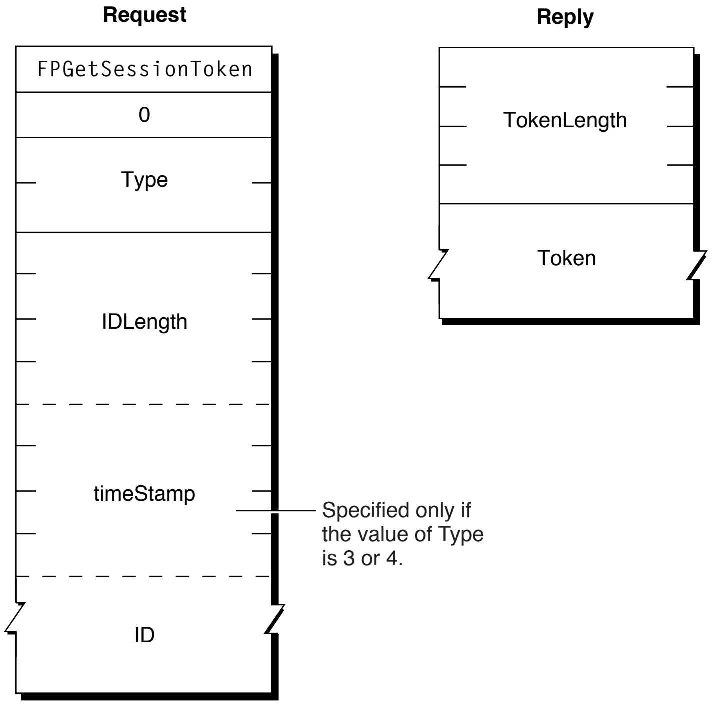
Table 3-30 Reply block for the `FPGetSessionToken` command

Name and size	Data
`TokenLength` (long)	Length of the token that follows.

Name and size	Data
Token (variable length)	Token that can be passed to FPDisconnectOldSession if the session is inadvertently disconnected and then re-established. If the client supports the Reconnect UAM, the token needs to be refreshed periodically, and is also sent to the server by the FPLoginExt (page 191) command to reconnect if the session is disconnected.

Figure 3-41 shows the request and reply blocks for the FPGetSessionToken command.

Figure 3-41 Request and reply blocks for the FPGetSessionToken command



FPGetSrvrInfo

Gets information about a server.

```
byte CommandCode
byte Pad
```

Parameter Descriptions

CommandCode

kFPGetSrvrInfo (15).

Pad

Pad byte.

`Result`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-32 (page 173) for the format of the reply block.

Discussion

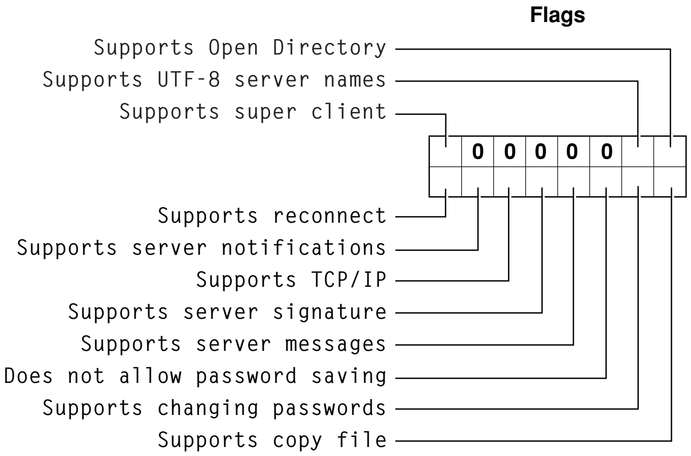
The reply block begins with the offset to the MachineType parameter, followed by the offset to the AFPVersions parameter, the offset to the UAMs parameter, and the offset to the VolumeIconAndMask parameter. The offsets are followed by the Flags parameter, the ServerName parameter padded to an even boundary,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

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

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

The Flags parameter indicates the server's support for certain features. If bit 9 in the Flags parameter is set, the reply block contains a UTF-8ServerNames offset to the server's name in UTF-8 format. Figure 3-42 shows how bits are used in the Flags parameter.

Figure 3-42 Bit usage in the Flags parameter


Flags
Supports Open Directory
Supports UTF-8 server names
Supports super client
0
0
0
0
0
Supports reconnect
Supports server notifications
Supports TCP/IP
Supports server signature
Supports server messages
Does not allow password saving
Supports changing passwords
Supports copy file

The AFPVersionsCount and UAMCount parameters are each one byte containing the number of AFP and UAM version strings that follow, with the strings packed back-to-back without padding. For the AFP versions supported by this version of AFP, see "AFP Version Strings" (page 258). For the UAMs supported by this version of AFP, see "AFP UAM Strings" (page 259).

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

The NetworkAddresses parameter contains the addresses that the client can use to connect to the server. Each address is stored as an AFP Network Address. The format of an AFP Network Address is shown in Figure 3-43.

Figure 3-43 AFP Network Address format

Length	Tag	Address

Each AFP Network Address consists of a length byte containing the total length in bytes of the Network Address, followed by a tag byte identifying the type of address the Address field contains, followed by up to 254 bytes of data. Table 3-31 lists the possible values of the `Length` and `Tag` fields and describes the type of address stored in the `Address` field.

Table 3-31 AFP Network Address fields

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

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

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

Table 3-32 describes the reply block for the `FPGetSrvrInfo command`.

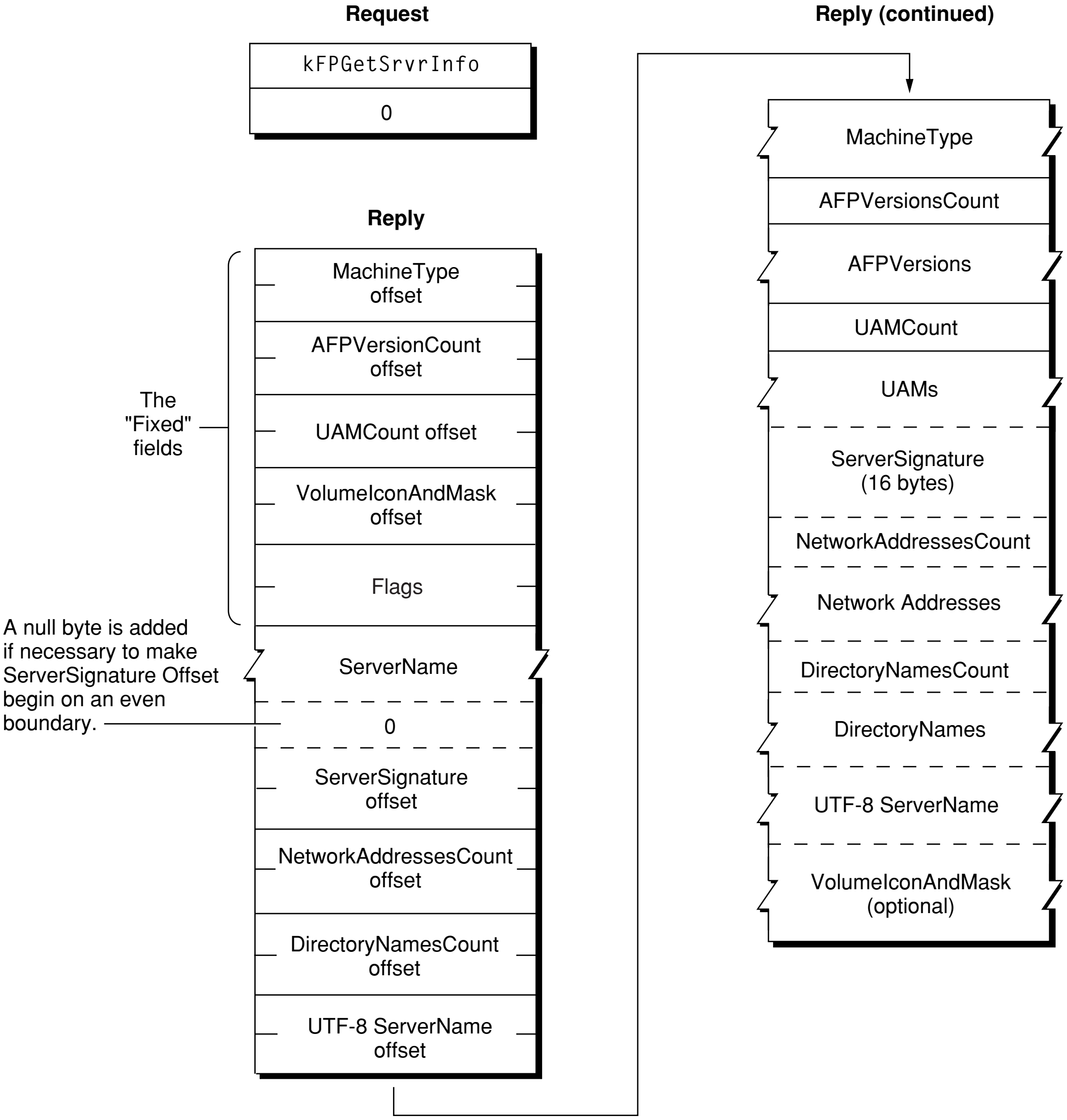
Table 3-32 Reply block for the FPGetSrvrInfo command

Name and size	Data
MachineType offset	Offset to the location in the reply block containing the server's machine type.
AFPVersionCount offset	Offset to the location in the reply block containing the number of AFP versions the server supports.
UAMCount offset	Offset to the location in the reply block containing the number of UAMs the server supports.
VolumeIconAndMask offset	Offset to the location in the reply block containing volume icon and mask data.
Flags (short)	Flags describing the server's capabilities. For bit definitions, see the section "Server Flags Bitmap" (page 256).
ServerName (string)	String containing the server's name.
ServerSignature offset	Offset to the location in the reply block containing the server's signature.
NetworkAddressesCount offset	Offset to the location in the reply block containing the number of AFP Network Addresses.
DirectoryNamesCount offset	Offset to the location in the reply block containing the number of Directory Names.
UTF-8ServerName	Offset to the location in the reply block containing the server's name in UTF-8 characters.
MachineType (string)	A string containing a description of the server's hardware, operating system, or both.
AFPVersionsCount	Number of AFP version strings that follow.
AFPVersions (packed string)	Each AFP version that the server supports in packed format. For each supported version, there is one byte stating the number of bytes in the version string that follows.
UAMsCount	Number of UAM strings that follow.
UAMs (packed string)	Each UAM that the server supports in packed format. For each supported UAM, there is one byte stating the number of bytes in the UAM name string that follows.
ServerSignature (16 bytes)	Sixteen-byte number that uniquely identifies the server, or zero if not supported. For AFP servers, supporting server signatures is optional, but AFP servers must provide a ServerSignature offset. An AFP server indicates that it supports server signatures by setting the kSupportsSrvrSig bit in the Flags parameter.

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

Figure 3-44 shows the request and reply blocks for the FPGetSrvrInfo command.

Figure 3-44 Request and reply blocks for the `FPGetSrvrInfo` command



FPGetSrvrMsg

Gets a message from a server.

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

Parameter Descriptions

`CommandCode`

`kFPGetSrvrMsg` (38).

`Pad`

Pad byte.

`MessageType`

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

`MessageBitmap`

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

`Result`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-33 (page 178) for the format of the reply block.

Discussion

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

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

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

There are two server message types:

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

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

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

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

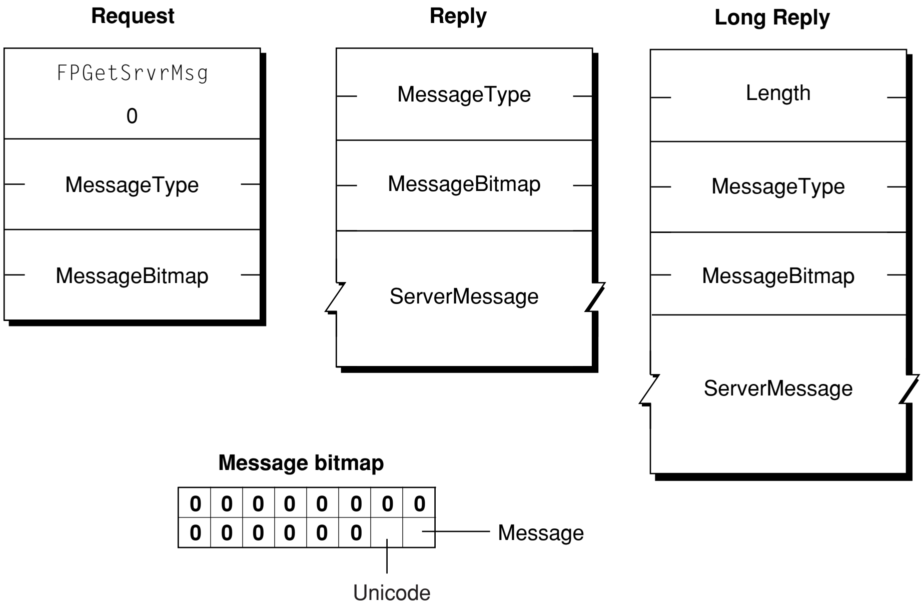
Table 3-33 describes the reply block for the FPGetSrvrMsg command.

Table 3-33 Reply block for the FPGetSrvrMsg command

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

Figure 3-45 shows the request and reply blocks for the FPGetSrvrMsg command.

Figure 3-45 Request and reply blocks for the `FPGetSrvrMsg` command



FPGetSrvrParms

Gets server parameters.

```
byte CommandCode
byte Pad
```

Parameter Descriptions

`CommandCode`

`kFPGetSrvrParms` (16).

`Pad`

Pad byte.

`Result`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-34 for the format of the reply block.

Discussion

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

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

For AFP 3.x, this command returns volume names in UTF8 format with a one byte length byte specifying any length up to 255. Note that the Finder limits setting volume names to no more than 27 characters.

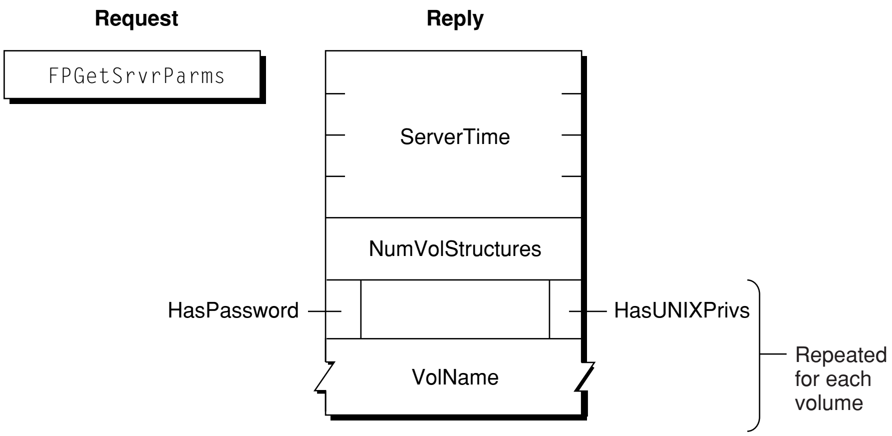
Table 3-34 describes the reply block for the FPGetSrvrParms command.

Table 3-34 Reply block for the FPGetSrvrParms command

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

Figure 3-46 shows the request and reply blocks for the FPGetSrvrParms command.

Figure 3-46 Request and reply blocks for the FPGetSrvrParms command



FPGetUserInfo

Gets information about a user.

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

Parameter Descriptions

CommandCode

kFPGetUserInfo (37).

Pad

Pad byte.

UserID

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

Bitmap

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

Result

kFPNoErr if no error occurred. See Table 3-35 for the other possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-36 (page 182) for the format of the reply block.

Discussion

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

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

Table 3-35 lists the result codes for the `FPGetUserInfo` command.

Table 3-35 Result codes for the `FPGetUserInfo` command

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

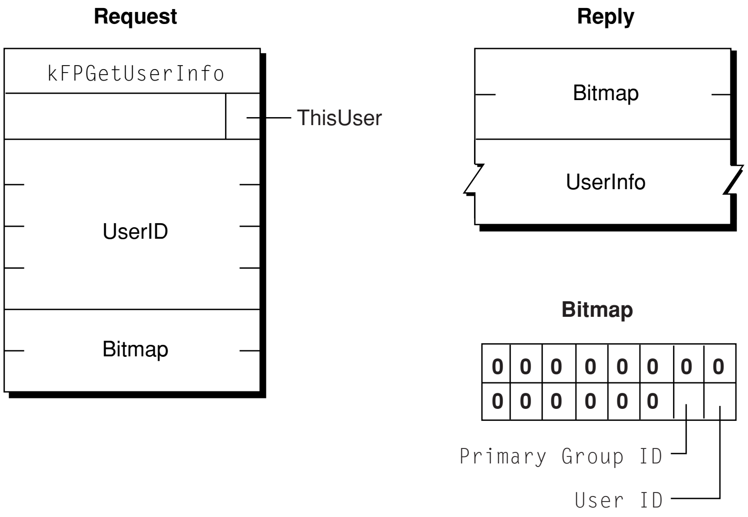
Table 3-36 describes the reply block for the `FPGetUserInfo` command.

Table 3-36 Reply block for the `FPGetUserInfo` command

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

Figure 3-47 shows the request and reply blocks for the `FPGetUserInfo` command.

Figure 3-47 Request and reply blocks for the `FPGetUserInfo` command



FPGetVolParms

Gets volume parameters.

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

Parameter Descriptions

CommandCode

kFPGetVolParms (17).

Pad

Pad byte.

VolumeID

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

Bitmap

Bitmap describing the parameters that are to be returned. Set the bit of the desired parameter. This bitmap is the same as the bitmap used by the FPOpenVol (page 209) command. For bit definitions for this bitmap, see "Volume Bitmap" (page 257).

Result

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

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-37 (page 184) for the format of the reply block.

Discussion

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

Before sending this command, you must call FPOpenVol (page 209) for the volume.

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

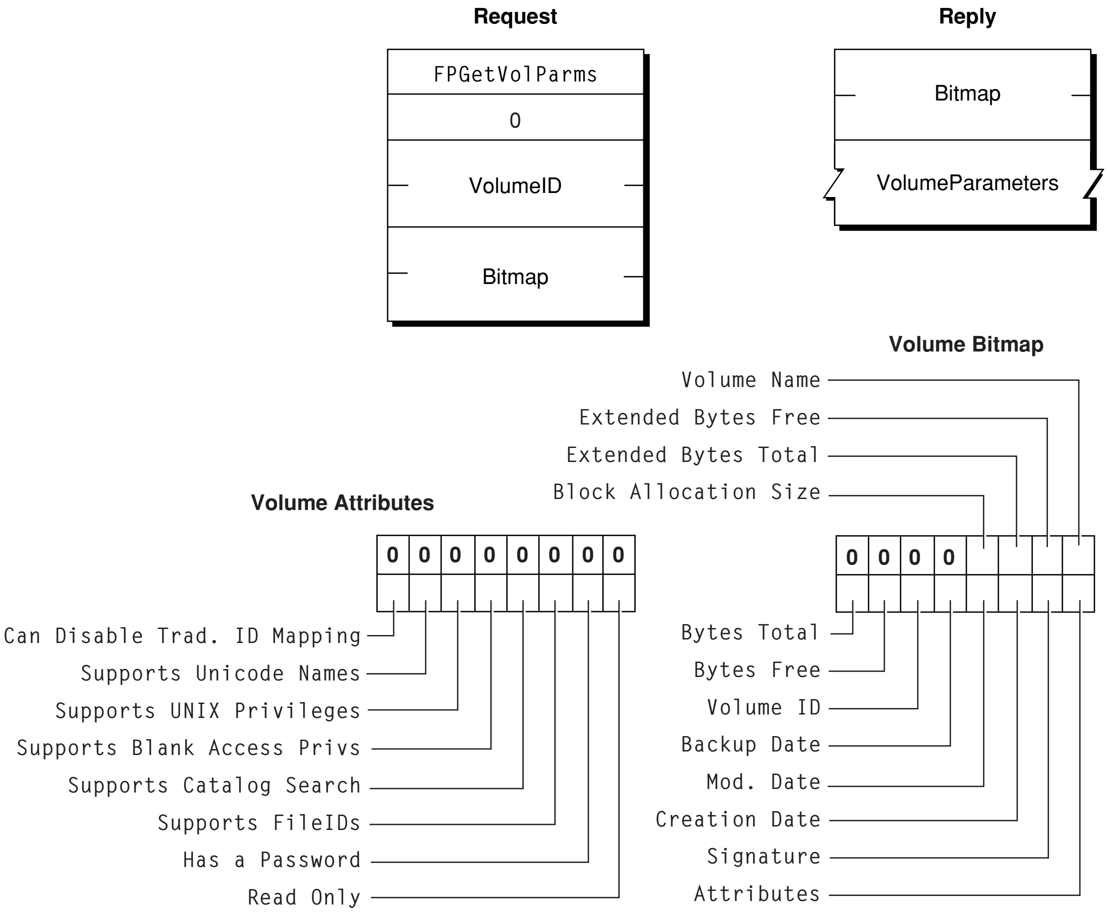
Table 3-37 describes the reply block for the FPGetVolParms command.

Table 3-37 Reply block for the FPGetVolParms command

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

Figure 3-48 shows the request and reply blocks for the FPGetVolParms command.

Figure 3-48 Request and reply blocks for the FPGetVolParms command



FPLogin

Establishes a session with a server.

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

Parameter Descriptions

`CommandCode`

kFPLogin (18).

`Pad`

Pad byte.

`AFPVersion`

String indicating which AFP version to use. For possible values, see "AFP Version Strings" (page 258).

`UAM`

String indicating which UAM to use. For possible values, see "`AFP UAM Strings`" (page 259).

`UserAuthInfo`

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

`Result`

`kFPNoErr` if no error occurred. See Table 3-38 (page 187) for the possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-39 (page 188) for the format of the reply block.

Discussion

This command establishes an AFP session with an AFP server. Before calling FPLogin, the AFP client should call FPGetAuthMethods (page 152) to obtain the AFP versions and UAMs that the server supports. From the list of AFP versions and UAMs returned by FPGetAuthMethods, the AFP client chooses the highest AFP version and the most secure UAM that the client supports and provides them as the AFPVersion and UAM parameters to the FPLogin command.

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

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

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

Table 3-38 lists the result codes for the FPLogin command.

Table 3-38 Result codes for the FPLogin command

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

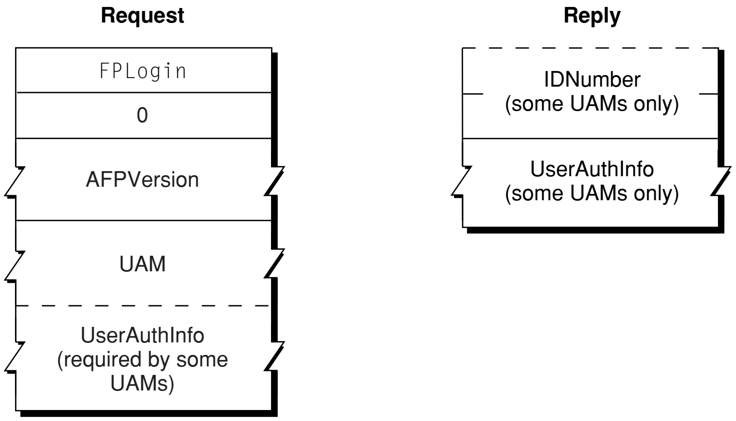
Table 3-39 describes the reply block for the FPLogin command.

Table 3-39 Reply block for the FPLogin command

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

Figure 3-49 shows the request and reply blocks for the FPLogin command.

Figure 3-49 Request and reply blocks for the FPLogin command


Request
FPLogin
0
AFPVersion
UAM
UserAuthInfo (required by some UAMs)
Reply
IDNumber (some UAMs only)
UserAuthInfo (some UAMs only)

FPLoginCont

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

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

Parameter Descriptions

CommandCode

kFPLoginCont (19).

Pad

Pad byte.

`ID`

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

`UserAuthInfo`

UAM-dependent information required to authenticate the user (can be null). The data type of `UserAuthInfo` depends on the UAM that was specified when `FPLogin` (page 185) or `FPLoginExt` (page 191) was called.

`Result`

`kFPNoErr` if no error occurred. See Table 3-40 (page 190) for the possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-41 (page 190) for the format of the reply block.

Discussion

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

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

Table 3-40 lists the result codes for the FPLoginCont command.

Table 3-40 Result codes for the FPLoginCont command

Result code	Explanation
kFPAuthContinue	Authentication is not yet complete.
kFPMiscErr	Non-AFP error occurred.
kFPNoServer	Server is not responding.
kFPParamErr	Authentication failed for an undisclosed reason.
kFPPwdExpiredErr	User's password has expired. User is required to change his or her password. The user is logged in but can only change his or her password or log out.
kFPPwdNeedsChangeErr	User's password needs to be changed. User is required to change his or her password. The user is logged on but can only change his or her password or log out.
kFPUserNotAuth	User was not authenticated because the password is incorrect.

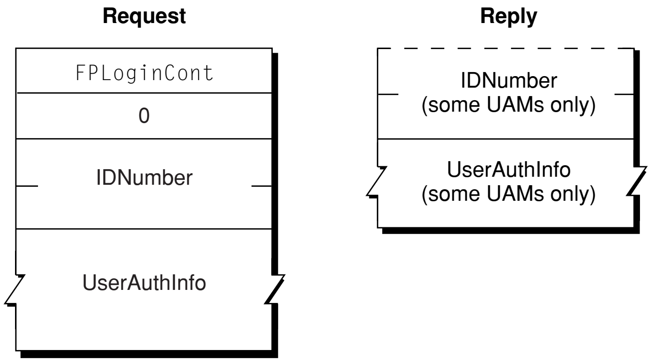
Table 3-41 describes the reply block for the FPLoginCont command.

Table 3-41 Reply block for the FPLoginCont command

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

Figure 3-50 shows the request and reply blocks for the FPLoginCont command.

Figure 3-50 Request and reply blocks for the FPLoginCont command



FPLoginExt

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

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

Parameter Descriptions

CommandCode

kFPLoginExt (63).

Pad

Pad byte.

Flags

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

AFPVersion

String indicating which AFP version to use. For possible values, see "AFP Version Strings" (page 258).

UAM

String indicating which UAM to use. For possible values, see "AFP UAM Strings" (page 259).

UserNameType

Type of name in UserName; always 3.

`UserName`

Name of the user, in AFPName format.

`PathType`

Type of names in `PathName`, where 1 indicate Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname for the Open Directory domain in which the user specified by `UserName` can be found. `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Pad`

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

`UserAuthInfo`

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

`Result`

`kFPNoErr` if no error occurred. See Table 3-42 (page 193) for other possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr` or `kFPAuthContinue`, the server returns a reply block. See Table 3-43 (page 194) for the format of the reply block.

Discussion

This command establishes an AFP session using the specified Open Directory domain in which information about the user can be found. Before sending this command, the AFP client should call FPGetAuthMethods (page 152) to obtain the UAMs that the Open Directory domain supports. From the list of UAMs returned by FPGetAuthMethods, the AFP client chooses the most secure UAM that it supports and provides it in the UAM parameter of the FPLoginExt command.

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

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

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

Table 3-42 lists the result codes for the FPLoginExt command.

Table 3-42 Result codes for the FPLoginExt command

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

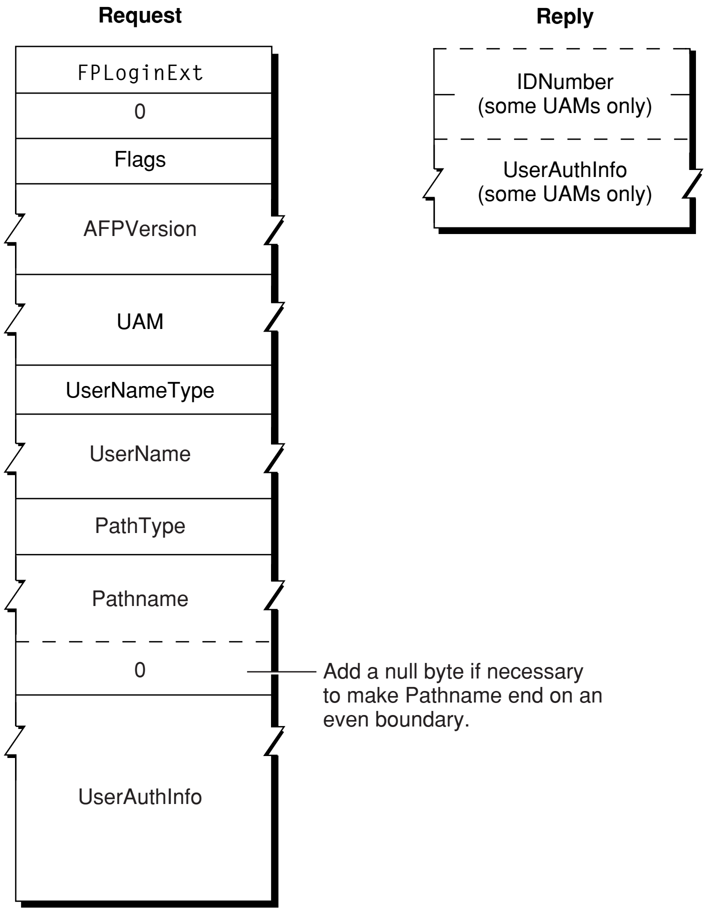
Table 3-43 describes the reply block for the FPLoginExt command.

Table 3-43 Reply block for the FPLoginExt command

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

Figure 3-51 shows the request and reply blocks for the FPLoginExt command.

Figure 3-51 Request and reply blocks for the FPLoginExt command



FPLogout

Terminates a session with a server.

```
byte CommandCode
byte Pad
```

Parameter Descriptions

CommandCode

kFPLogout (20).

Pad

Pad byte.

Result

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

ReplyBlock

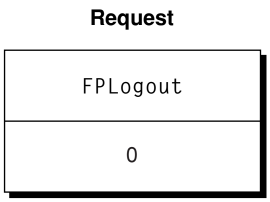
None.

Discussion

This command terminates sessions established by FPLogin (page 185) and FPLoginExt (page 191). The server flushes and closes any forks opened by the session, frees all session-related resources, and invalidates the session reference number.

Figure 3-52 shows the request block for the FPLogout command.

Figure 3-52 Request block for the FPLogout command



FPMapID

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

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

Parameter Descriptions

CommandCode

kFPMapID (21).

Subfunction

Subfunction code, where 1 maps a User ID to a Macintosh Roman user name, 2 maps a Group ID to a Macintosh Roman group name, 3 maps a User ID to a Unicode user name, and 4 maps a Group ID to a Unicode group name.

ID

Group ID or User ID that is to be mapped.

Result

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

ReplyBlock

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

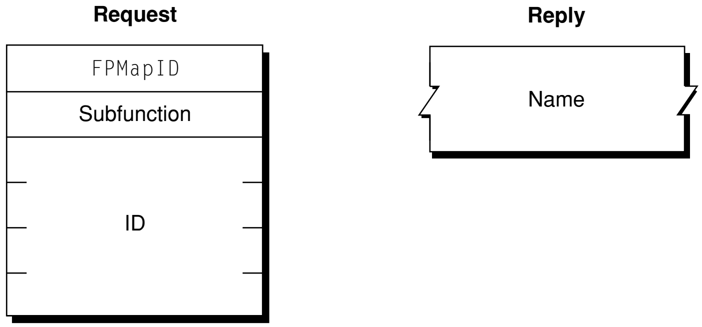
Discussion

The server retrieves a user or group name in Macintosh Roman or Unicode format corresponding to the specified User ID or Group ID.

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

Figure 3-53 shows the request and reply blocks for the FPMapID command.

Figure 3-53 Request and reply blocks for the FPMapID command



FPMapName

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

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

Parameter Descriptions

`CommandCode`

`kFPMapName` (22).

`Subfunction`

Subfunction code, where 1 maps a Unicode user name to a User ID, 2 maps a Unicode group name to a Group ID, 3 maps a Macintosh Roman user name to User ID, and 4 maps a Macintosh Roman group name to a Group ID.

`Name`

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

`Result`

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

`ReplyBlock`

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

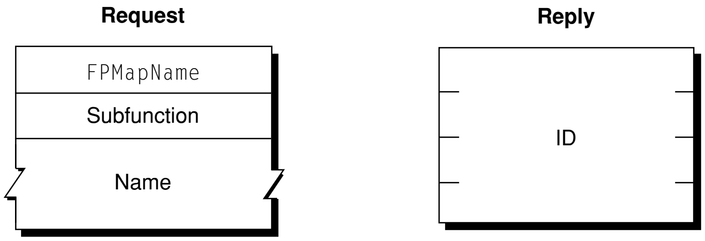
Discussion

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

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

Figure 3-54 shows the request and reply blocks for the `FPMapName` command.

Figure 3-54 Request and reply blocks for the `FPMapName` command



FPMoveAndRename

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

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

```
long SourceDirectoryID
long DestDirectoryID
byte SourcePathType
string SourcePathname
byte DestPathType
string DestPathname
byte NewType
string NewName
```

Parameter Descriptions

`CommandCode`

`kFPMoveAndRename` (23).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`SourceDirectoryID`

Source ancestor Directory ID.

`DestDirectoryID`

Destination ancestor Directory ID.

`SourcePathType`

Type of names in SourcePathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`SourcePathname`

Pathname of the file or directory to be moved (may be null if a directory is being moved). `SourcePathname` is a string if `SourcePathType` is 1 or 2, or an AFPName if `SourcePathType` is 3.

`DestPathType`

Type of names in DestPathname, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`DestPathname`

Pathname of the file or directory to be moved (may be null if a directory is being moved). DestPathname is a string if SourcePathType is 1 or 2, or an AFPName if SourcePathType is 3.

`NewType`

Type of name in `NewName`, where 1 indicates Short Name, 2 indicates Long Name, and 3 indicates Unicode name.

`NewName`

New name of file or directory (may be null). `NewName` is a string if `NewType` is 1 or 2, or an AFPName if `NewType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-44 (page 200) for other possible result codes.

`ReplyBlock`

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

Discussion

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

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

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

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

The user must have previously called FPOpenVol (page 209) for this volume.

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

Table 3-44 lists the result codes for the FPMoveAndRename command.

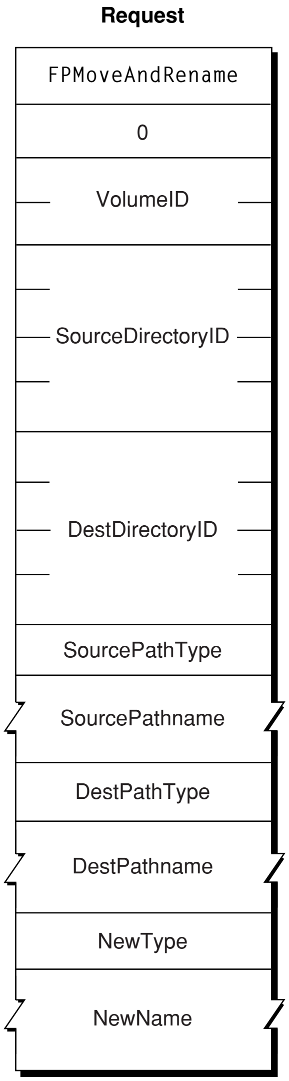
Table 3-44 Result codes for the FPMoveAndRename command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to move or rename the specified file or directory.
kFPCantMove	Attempt was made to move a directory into one of its descendent directories.
kFPInsideSharedErr	Directory being moved contains a share point and is being moved into a directory that is shared or is the descendent of a directory that is shared.
kFPInsideTrashErr	Shared directory is being moved into the Trash; a directory is being moved to the trash and it contains a shared folder.
kFPMiscErr	Non-AFP error occurred.
kFPObjectExists	File or directory having the name specified by NewName already exists.
kFPObjectLocked	Directory being moved, renamed, or moved and renamed is marked RenameInhibit; file being moved and renamed is marked RenameInhibit.

Result code	Explanation
kFPObjectNotFound	Input parameters do not point to an existing file or directory.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown; a pathname or NewName is invalid.
kFPVolLocked	Volume is ReadOnly.

Figure 3-55 shows the request block for the FPMoveAndRename command.

Figure 3-55 Request block for the FPMoveAndRename command



FPOpenDir

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

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

Parameter Descriptions

`CommandCode`

`kFPOpenDir` (25).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname of the file or directory to be moved (may be null if a directory is being moved). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-45 (page 203) for other possible result codes.

`ReplyBlock`

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

Discussion

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

Although this command can obtain a Directory ID for a directory on a fixed Directory ID volume, the recommended way to obtain a Directory ID for a directory on a fixed Directory ID volume is to call FPGetFileDirParms (page 156).

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

The user must have previously called FPOpenVol (page 209) for this volume.

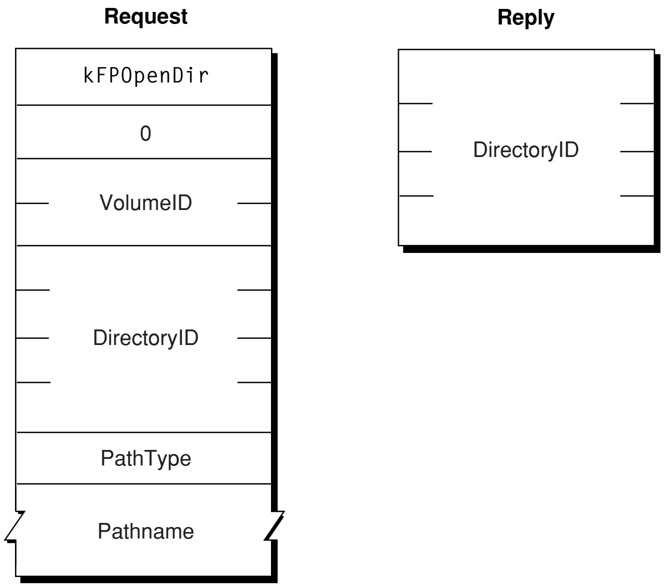
Table 3-45 lists the result codes for the FPOpenDir command.

Table 3-45 Result codes for the FPOpenDir command

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

Figure 3-56 shows the request and reply blocks for the FPOpenDir command.

Figure 3-56 Request and reply blocks for the FPOpenDir command



FPOpenDT

Opens the Desktop database on a particular volume.

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

Parameter Descriptions

CommandCode

kFPOpenDT (48).

Pad

Pad byte.

VolumeID

Volume ID.

Result

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

ReplyBlock

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

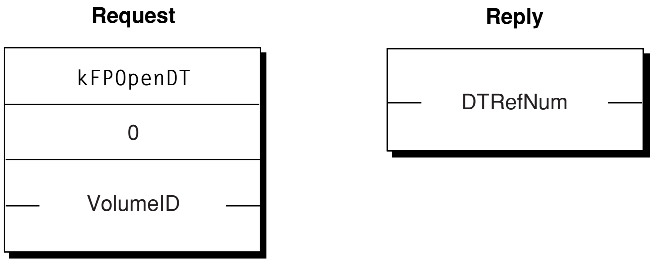
Discussion

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

The user must have previously called FPOpenVol (page 209) for this volume.

Figure 3-57 shows the request and reply blocks for the FPOpenDT command.

Figure 3-57 Request and reply blocks for the FPOpenDT command



FPOpenFork

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

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

Parameter Descriptions

CommandCode

kFPOpenFork (26).

Flag

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

VolumeID

Volume ID.

DirectoryID

Ancestor Directory ID.

`Bitmap`

Bitmap describing the fork parameters to be returned. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the `FPGetFileDirParms` (page 156) command and can be null. For bit definitions for the File bitmap, see "File Bitmap" (page 253).

`AccessMode`

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

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the desired file (cannot be null). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-46 (page 207) for the possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-47 (page 208) for the format of the reply block.

Discussion

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

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

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

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

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

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

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

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

The user must have previously called `FPOpenVol` (page 209) for this volume. Each fork must be opened separately; a unique fork reference is returned for each fork.

Table 3-46 lists the result codes for the `FPOpenFork` command.

Table 3-46 Result codes for the `FPOpenFork` command

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

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

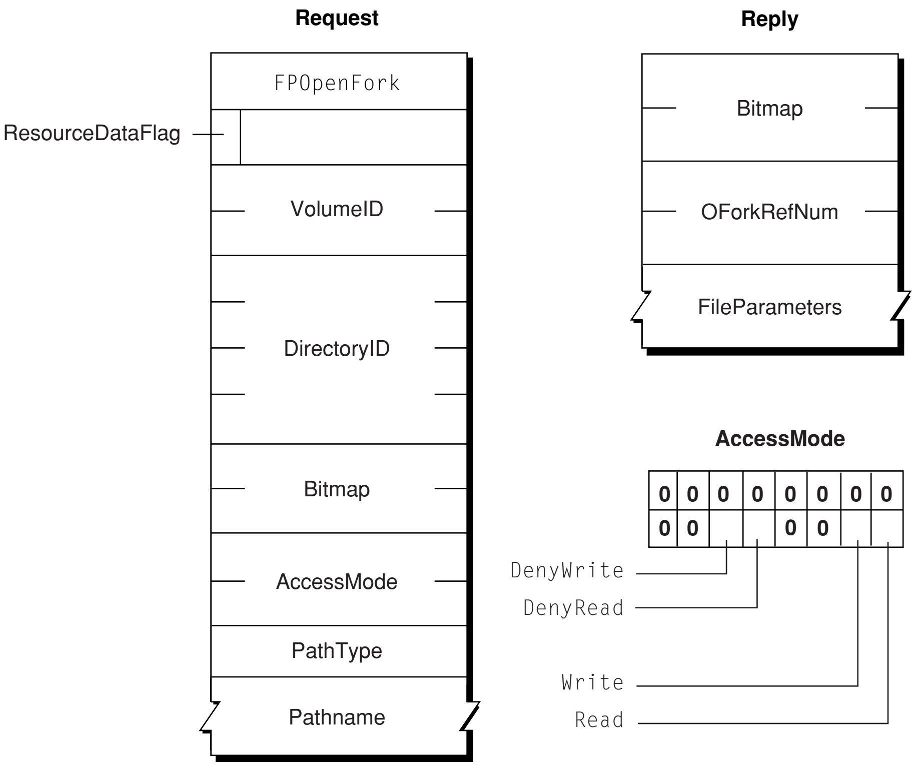
Table 3-47 describes the reply block for the FPOpenFork command.

Table 3-47 Reply block for the FPOpenFork command

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

Figure 3-58 shows the request and reply blocks for the FPOpenFork command.

Figure 3-58 Request and reply blocks for the `FPOpenFork` command



FPOpenVol

Opens a volume.

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

Parameter Descriptions

`CommandCode`

`kFPOpenVol` (24).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`Bitmap`

Bitmap describing the parameters that are to be returned. Set the bit that corresponds to each desired parameter. The bitmap is the same as the Volume bitmap used by the `FPGetVolParms` (page 183) command and cannot be null. For bit definitions, see "Volume Bitmap" (page 257).

`VolumeName`

Name of the volume as returned by `FPGetSrvrParms` (page 179).

`Password`

Optional volume password.

`Result`

`kFPNoErr` if no error occurred. See Table 3-48 (page 211) for the possible result codes.

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block. See Table 3-49 (page 211) for the format of the reply block.

Discussion

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

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

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

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

Table 3-48 lists the result codes for the FPOpenVol command.

Table 3-48 Result codes for the FPOpenVol command

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

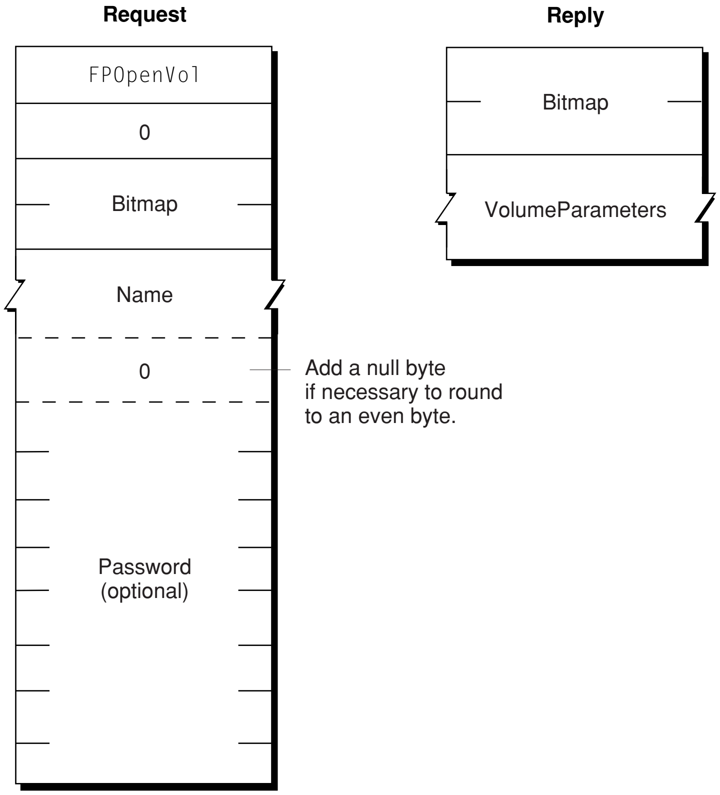
Table 3-49 describes the reply block for the FPOpenVol command.

Table 3-49 Reply block for the FPOpenVol command

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

Figure 3-59 shows the request and reply blocks for the FPOpenVol command.

Figure 3-59 Request and reply blocks for the FPOpenVol command



FPRead

Reads a block of data.

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

Parameter Descriptions

`CommandCode`

  `kFPRead` (27).

`Pad`

  Pad byte.

`OForkRefNum`

  Open fork reference number.

`Offset`

  Number of the first byte to read.

`ReqCount`

Number of bytes to read.

`NewLineMask`

Mask for determining where the read should terminate.

`NewLineChar`

Character for determining where the read should terminate.

`Result`

`kFPNoErr` if no error occurred. See Table 3-50 (page 214) for the possible result codes.

`ActualCount`

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

`ReplyBlock`

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

Discussion

This command retrieves the specified range of bytes from an open fork. Call FPOpenFork (page 205) to open the fork. The server begins reading at the byte number specified by the Offset parameter. Reading stops when one of the following occur:

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

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

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

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

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

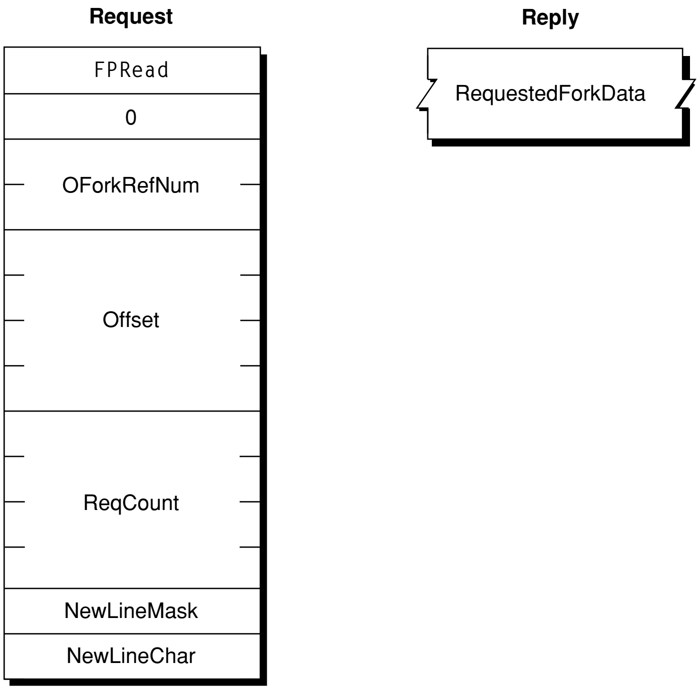
Table 3-50 (page 214) lists the result codes for the FPRead command.

Table 3-50 Result codes for the FPRead command

Result code	Explanation
kFPAccessDenied	Fork was not opened for read access.
kFPEOFErr	End of fork was reached.
kFPLockErr	Some or all of the requested range is locked by another user.
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or open fork reference number is unknown; ReqCount or Offset is negative; NewLineMask is invalid.

Figure 3-60 shows the request and reply blocks for the FPRead command.

Figure 3-60 Request and reply blocks for the FPRead command



FPReadExt

Reads a block of data.

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

Parameter Descriptions

`CommandCode`

`kFPReadExt` (60).

`Pad`

Pad byte.

`OForkRefNum`

Open fork reference number.

`Offset`

Number of the first byte to read.

`ReqCount`

Number of bytes to read.

`Result`

`kFPNoErr` if no error occurred. See Table 3-51 (page 217) for the possible result codes.

`ActualCount`

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

`ReplyBlock`

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

Discussion

This command retrieves the specified range of bytes from an open fork. Call FPOpenFork (page 205) to open the fork.

This command differs from the FPRead (page 212) command in that this command is prepared to handle large values that may be returned for files the reside in volumes larger than 4 GB in size. Also, this command does not support the NewlineMask and NewlineChar parameters that FPRead supports.

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

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

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

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

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

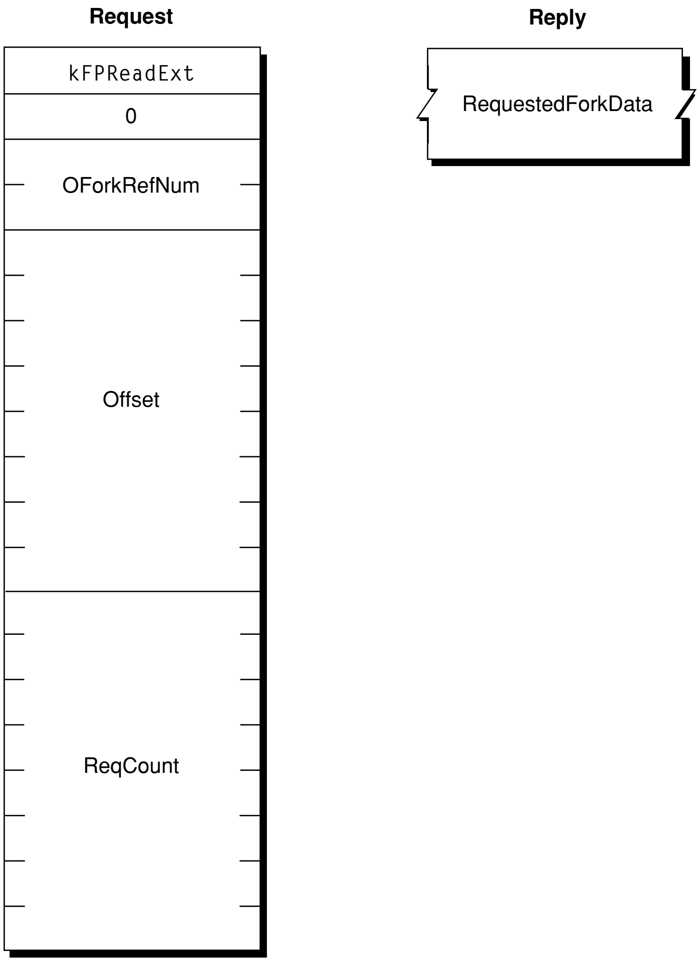
Table 3-51 lists the result codes for the FPReadExt command.

Table 3-51 Result codes for the FPReadExt command

Result code	Explanation
kFPAccessDenied	Fork was not opened for read access.
kFPEOFErr	End of fork was reached.
kFPLockErr	Some or all of the requested range is locked by another user.
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or open fork reference number is unknown; ReqCount or Offset is negative.

Figure 3-61 shows the request and reply blocks for the FPReadExt command.

Figure 3-61 Request and reply blocks for the FPReadExt command



FPRemoveAPPL

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

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

Parameter Descriptions

`CommandCode`

`kFPRemoveAPPL` (54).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`DirectoryID`

Ancestor Directory ID.

`FileCreator`

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

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the desired file (cannot be null). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-52 (page 220) for the possible result codes.

`ReplyBlock`

None.

Discussion

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

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

The user must have previously called `FPOpenDT` (page 204) for the corresponding volume. In addition, the file must exist in the specified directory before this command is sent.

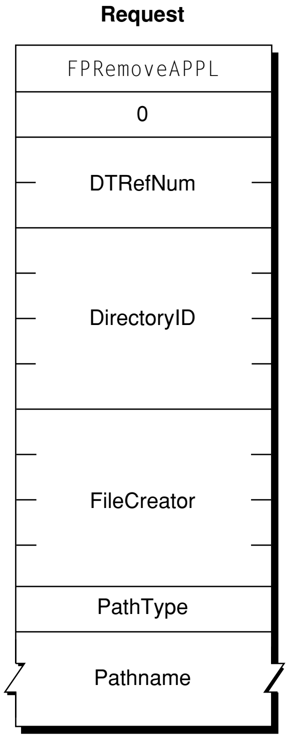
Table 3-52 lists the result codes for the `FPRemoveAPPL` command.

Table 3-52 Result codes for the `FPRemoveAPPL` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPItemNotFound`	No APPL mapping corresponding to the input parameters was found in the Desktop database.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file.
`kFPParamErr`	Session reference number or Desktop database reference number is unknown.

Figure 3-62 shows the request and reply blocks for the `FPRemoveAPPL` command.

Figure 3-62 Request and reply blocks for the `FPRemoveAPPL` command



FPRemoveComment

Removes a comment from a volume's Desktop database.

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

Parameter Descriptions

`CommandCode`

`kFPRemoveComment` (57).

`Pad`

Pad byte.

`DTRefNum`

Desktop database reference number.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the CNode whose comment is being removed (cannot be null). `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`Result`

`kFPNoErr` if no error occurred. See Table 3-53 (page 223) for the possible result codes.

`ReplyBlock`

None.

Discussion

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

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

The user must have previously called `FPOpenDT` (page 204) for the corresponding volume.

Table 3-53 lists the result codes for the `FPRemoveComment` command.

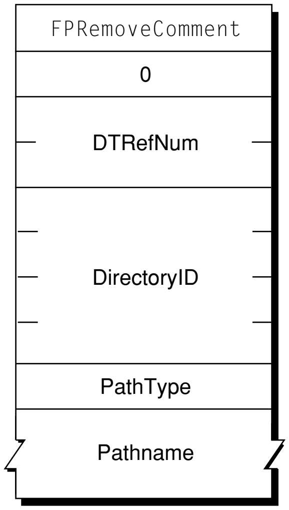
Table 3-53 Result codes for the `FPRemoveComment` command

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

Figure 3-63 shows the request and reply blocks for the `FPRemoveComment` command.

Figure 3-63 Request and reply blocks for the `FPRemoveComment` command

Request



FPRename

Renames a file or directory.

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

Parameter Descriptions

`CommandCode`

kFPRename (28).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`PathType`

Type of names in `Pathname`, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

`Pathname`

Pathname to the CNode whose comment is being removed (cannot be null). `Pathname` is a string if PathType is 1 or 2, or an AFPName if `PathType` is 3.

NewType

Type of names in NewName, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

NewType

Type of names in NewName, where 1 indicates Short Names, 2 indicates Long Names, and 3 indicates Unicode names.

Result

kFPNoErr if no error occurred. See Table 3-54 (page 226) for the possible result codes.

ReplyBlock

None.

Discussion

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

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

The user must have previously called FPOpenVol (page 209) for the corresponding volume.

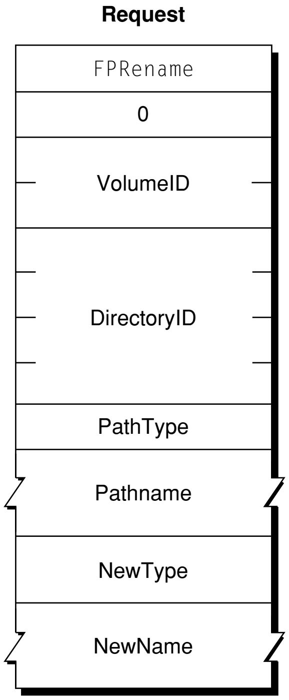
Table 3-54 lists the result codes for the FPRename command.

Table 3-54 Result codes for the FPRename command

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

Figure 3-64 shows the request block for the FPRename command.

Figure 3-64 Request block for the FPRename command



FPResolveID

Gets parameters for a file by File ID.

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

Parameter Descriptions

CommandCode

kFPResolveID (41).

Pad

Pad byte.

VolumeID

Volume ID.

FileID

File ID to be resolved.

Bitmap

Bitmap describing the parameters to return. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap parameter of the FPGetFileDirParms (page 156) command. For bit definitions for the this bitmap, see "File Bitmap" (page 253).

Result

kFPNoErr if no error occurred. See Table 3-55 (page 229) for the possible result codes.

ReplyBlock

If the result code is kFPNoErr, the server returns a reply block. See Table 3-56 (page 229) for the format of the reply block.

Discussion

The parameters returned by this command can be any parameter specified in the `FPGetFileDirParms` (page 156) command.

Before sending this command, call `FPOpenVol` (page 209) to open the volume on which the file represented by `FileID` resides.

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

Table 3-55 lists the result codes for the `FPResolveID` command.

Table 3-55 Result codes for the `FPResolveID` command

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

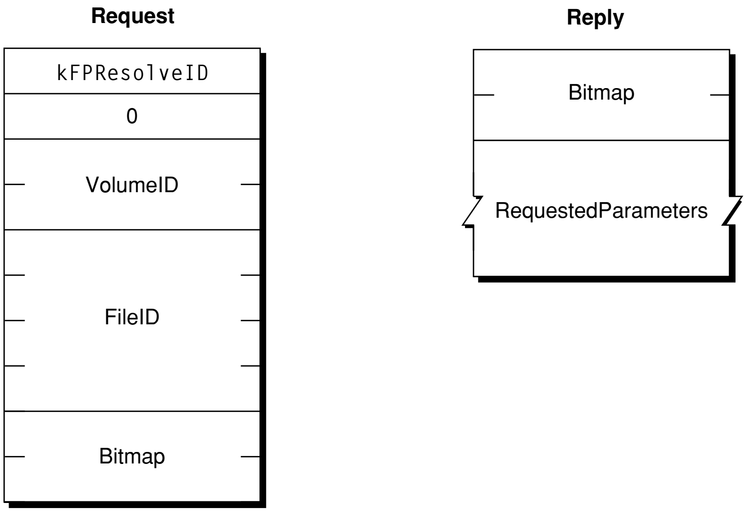
Table 3-56 describes the reply block for the `FPResolveID` command.

Table 3-56 Reply block for the `FPResolveID` command

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

Figure 3-64 shows the request and reply blocks for the `FPResolveID` command.

Figure 3-65 Request and reply blocks for the `FPResolveID` command



FPSetDirParms

Sets parameters for a directory.

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

Parameter Descriptions

`CommandCode`

`kFPSetDirParms` (29).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`Bitmap`

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap is the same as the `DirectoryBitmap` parameter of the `FPGetFileDirParms` (page 156) command. For bit definitions for this bitmap, see "Directory Bitmap" (page 251).

`PathType`

Type of name in `Pathname`, where 1 indicates Short Name, 2 indicates Long Name, and 3 indicates Unicode name.

`Pathname`

Pathname to the desired directory. `Pathname` is a string if PathType is 1 or 2, or an AFPName if `PathType` is 3.

`DirectoryParameters`

Parameters to be set, packed in bitmap order.

`Result`

`kFPNoErr` if no error occurred. See Table 3-57 (page 232) for the possible result codes.

`ReplyBlock`

None.

Discussion

This command sets or clears certain parameters and attributes that are common to both files and directories. The parameters are the Invisible and System attributes, Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters, such as Long Name and Short Name, must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

Changing a directory's access rights immediately affects other open sessions. If the user does not have the access rights to set one of the parameters, a `kFPAccessDenied` result code is returned and no parameters are set.

To set a directory's access privileges, Owner ID, Group ID, or to change the DeleteInhibit, RenameInhibit, WriteInhibit, or Invisible attributes, the user must have search or write access to all ancestors, including this directory's parent directory, and the user must be the owner of the directory. To set any parameter other than the ones mentioned above for an empty directory, the user must have search or write access to all ancestors, except the parent directory, as well as write access to the parent directory. To set any parameter other than the ones mentioned above for a directory that is not empty, the user must have search access to all ancestors, including the parent directory, as well as write access to the parent directory.

The user must have previously called `FPOpenVol` (page 209) for this volume.

This command cannot be used to set a directory's name; instead, use `FPRename` (page 224). This command cannot be used to set a directory's Parent Directory ID; instead, use `FPMoveAndRename` (page 197). This command cannot be used to set a directory's Directory ID or Offspring Count.

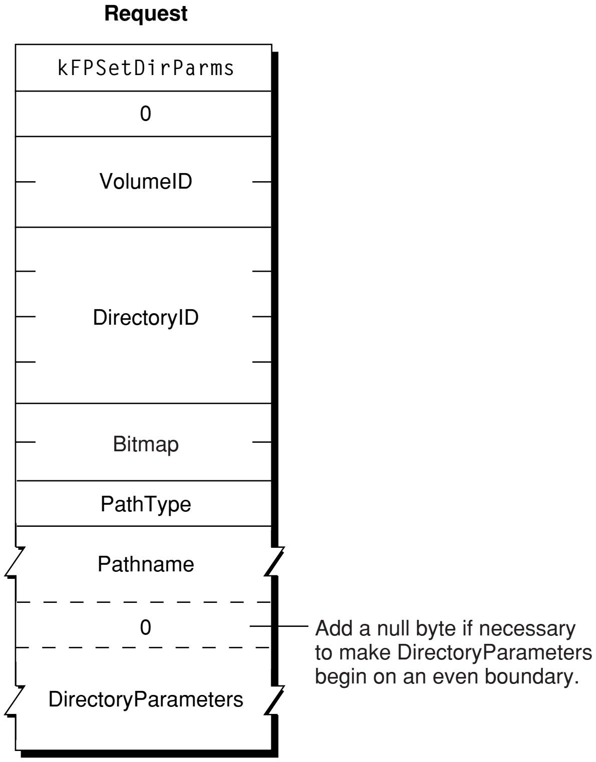
Table 3-57 lists the result codes for the `FPSetDirParms` command.

Table 3-57 Result codes for the `FPSetDirParms` command

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

Figure 3-66 shows the request block for the FPSetDirParms command.

Figure 3-66 Request block for the FPSetDirParms command



FPSetFileDirParms

Sets parameters for a file or a directory.

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

Parameter Descriptions

CommandCode

kFPSetFileDirParms (35).

Pad

Pad byte.

`VolumeID`

Volume ID.

`DirectoryID`

Ancestor Directory ID.

`Bitmap`

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap can be the same as the `DirectoryBitmap` or the `FileBitmap` parameter of the `FPGetFileDirParms` (page 156) command, but this command can only set the parameters that are common to both bitmaps. For bit definitions for the Directory and bitmap, see "Directory Bitmap" (page 251); for bit definitions for the File bitmap, see "File Bitmap" (page 253).

`PathType`

Type of name in `Pathname`, where 1 indicates Short Name, 2 indicates Long Name, and 3 indicates Unicode name.

`Pathname`

Pathname to the desired file or directory. `Pathname` is a string if `PathType` is 1 or 2, or an AFPName if `PathType` is 3.

`FileDirParameters`

Parameters to be set, packed in bitmap order.

`Result`

`kFPNoErr` if no error occurred. See Table 3-58 (page 235) for the possible result codes.

`ReplyBlock`

None.

Discussion

This command sets or clears certain parameters and attributes that are common to both files and directories. The parameters are the Invisible and System attributes, Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters, such as Long Name and Short Name, must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

If necessary, a null byte must be added between `Pathname` and `DirectoryParameters` in the request block to make `DirectoryParameters` begin on an even boundary.

If the Attributes parameter is included, the Set/Clear bit indicates that the specified attributes are to be set (1) or cleared (0). Therefore, it is not possible to set some attributes and clear other attributes in the same command.

If this command changes the CNode's attributes or sets the CNode's dates (except modification date), Finder Info, or UNIX privileges, the modification date of the CNode is set to the server's clock. If this command changes the CNode’s Invisible attribute, the modification date of the CNode’s parent directory is set to the server’s clock.

To set the parameters for a directory that is not empty, the user needs search access to all ancestors, including the parent directory, as well as write access to the parent directory. To set parameters for an empty directory, the user needs search or write access to all ancestors, except the parent directory, as well as write access to the parent directory.

To set parameters for a file that is not empty, the user needs search access to all ancestors, except the parent directory, as well as write access to the parent directory. To set parameters for an empty file, the user needs search or write access to all ancestors, except the parent directory, as well as write access to the parent directory.

The user must have previously called `FPOpenVol` (page 209) for this volume.

For files, call `FPSetFileParms` (page 236) to set parameters and attributes that `FPSetFileDirParms` cannot set. For directories, call `FPSetDirParms` (page 230) to set parameters and attributes that `FPSetFileDirParms` cannot set.

Table 3-58 lists the result codes for the `FPSetFileDirParms` command.

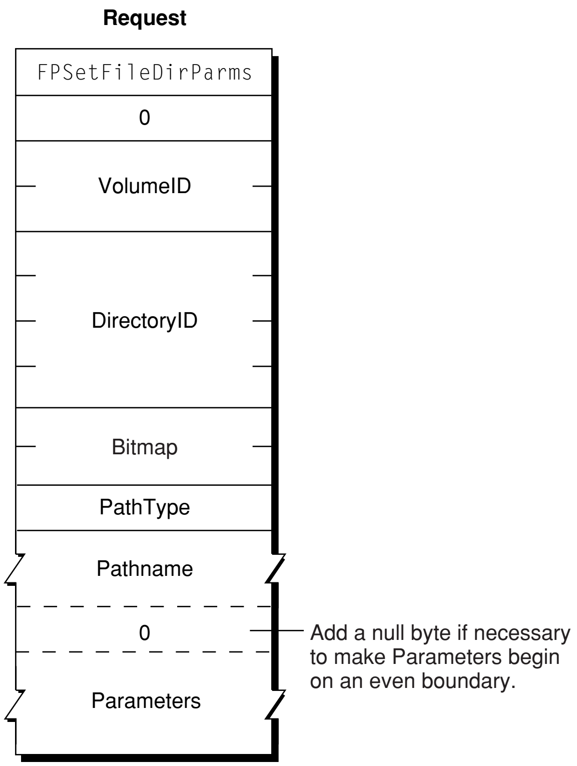
Table 3-58 Result codes for the `FPSetFileDirParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.

Result code	Explanation
kFPObjectNotFound	Input parameters do not point to an existing file or directory.
kFPParamErr	Session reference number, Volume ID, or pathname type is unknown; pathname is invalid.
kFPVolLocked	Volume is ReadOnly.

Figure 3-67 shows the request block for the FPSetFileDirParms command.

Figure 3-67 Request block for the FPSetFileDirParms command



FPSetFileParms

Sets parameters for a file.

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

Parameter Descriptions

`CommandCode`

`kFPSetFileParms` (30).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`VolumeID`

Ancestor Directory ID.

`Bitmap`

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap can be the same as the FileBitmap parameter of the `FPGetFileDirParms` (page 156) command. For bit definitions for the Directory bitmap, see "Directory Bitmap" (page 251); for bit definitions for the File bitmap, see "File Bitmap" (page 253).

`PathType`

Type of name in `Pathname`, where 1 indicates Short Name, 2 indicates Long Name, and 3 indicates Unicode name.

`Pathname`

Pathname to the desired file or directory. `Pathname` is a string if PathType is 1 or 2, or an AFPName if `PathType` is 3.

`FileParameters`

Parameters to be set, packed in bitmap order.

`Result`

`kFPNoErr` if no error occurred. See Table 3-59 (page 238) for the possible result codes.

`ReplyBlock`

None.

Discussion

The parameters must be packed, in bitmap order, in the request block. Variable-length parameters must be kept at the end of the block. To do this, variable-length parameters are represented in bitmap order as fixed-length offsets (integers). These offsets are measured from the start of the parameters to the start of the variable-length parameters. The actual variable-length parameters are then packed after all fixed-length parameters.

If necessary, a null byte must be added between `Pathname` and `FileParameters` in the request block to make `FileParameters` begin on an even boundary.

The following parameters may be set or cleared: Attributes (all attributes except DAlreadyOpen, RAlreadyOpen, and CopyProtect), Creation Date, Modification Date, Backup Date, Finder Info, and UNIX privileges.

If the Attributes parameter is included, the Set/Clear bit indicates that the specified attributes are to be set (1) or cleared (0). Therefore, it is not possible to set some attributes and clear other attributes in the same call.

If this command changes the file's Invisible attribute, the modification date of the file's parent directory is set to the server's clock. If this command changes the file's Attributes or sets any dates (except modification date), or Finder Info, the file's modification date is set to the server's clock.

If the file is empty (both forks are zero length), the user must have search or write access to all ancestors, except this file's parent directory, as well as write access to the parent directory. If either fork is not empty, the user must have search access to all ancestors except the parent directory, as well as read and write access to the parent directory.

The user must have previously called `FPOpenVol` (page 209) for this volume.

This command cannot be used to set a file's name; instead, use `FPRename` (page 224). This command cannot be used to set the file's Parent Directory ID; instead, use `FPMoveAndRename` (page 197). This command cannot be used to set a file's fork lengths; instead, call `FPSetForkParms` (page 239). This command cannot be used to set a file's Node ID.

Table 3-59 lists the result codes for the `FPSetFileParms` command.

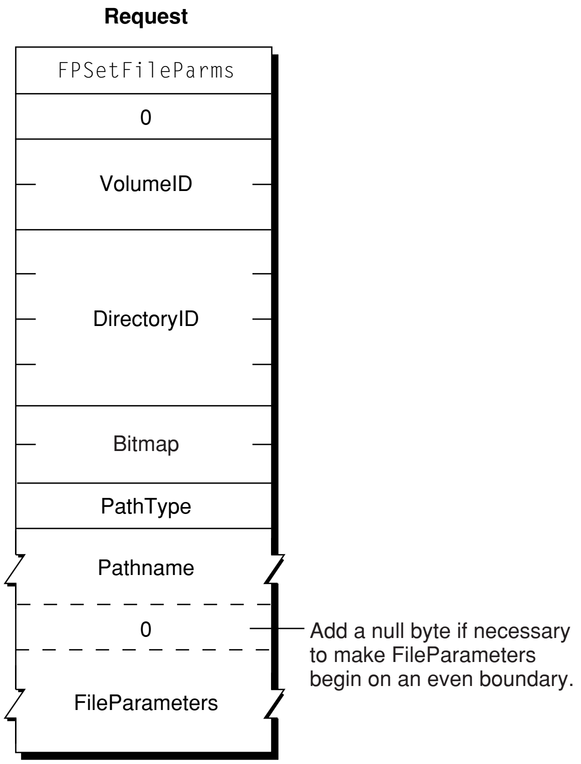
Table 3-59 Result codes for the `FPSetFileParms` command

Result code	Explanation
`kFPAccessDenied`	User does not have the access privileges required to use this command.
`kFPBitmapErr`	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
`kFPMiscErr`	Non-AFP error occurred.
`kFPObjectNotFound`	Input parameters do not point to an existing file.
`kFPObjectTypeErr`	Input parameters point to a directory.

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

Figure 3-68 shows the request block for the FPSetFileParms command.

Figure 3-68 Request block for the FPSetFileParms command


Request
FPSetFileParms
0
VolumeID
DirectoryID
Bitmap
PathType
Pathname
0
FileParameters
Add a null byte if necessary to make FileParameters begin on an even boundary.

FPSetForkParms

Sets the length of a fork.

```
byte CommandCode
byte Pad
short OForkRefNum
short Bitmap
long ForkLen
```

Parameter Descriptions

CommandCode

kFPSetForkParms (31).

Pad

Pad byte.

OForkRefNum

Open fork reference number.

Bitmap

Bitmap describing the parameters to set. Set the bit that corresponds to each desired parameter. This bitmap is the same as the FileBitmap of the FPGetFileDirParms (page 156) command, but only the Data Fork Length, Resource Fork Length, Extended Data Fork Length, and Extended Resource Fork Length parameters can be set. For bit definitions for this bitmap, see "File Bitmap" (page 253).

ForkLen

New end-of-fork value.

Result

kFPNoErr if no error occurred. See Table 3-60 (page 241) for the possible result codes.

ReplyBlock

None.

Discussion

The Bitmap and ForkLen parameters are passed to the server, which changes the length of the fork specified by OForkRefNum. The server returns a kFPBitmapErr result code if the command tries to set the length of the file's other fork or if it tries to set any other file parameter.

The server returns a kFPLockErr result code if an attempt is made to truncate the fork in a way that would eliminate a range or part of a range that is locked by another user.

The fork must be open for writing by the user.

This command cannot set a file's name; instead, use FPRename (page 224). This command cannot set a file's Parent Directory ID; instead, use FPMoveAndRename (page 197). This command cannot set a file's file number.

Table 3-60 lists the result codes for the FPSetForkParms command.

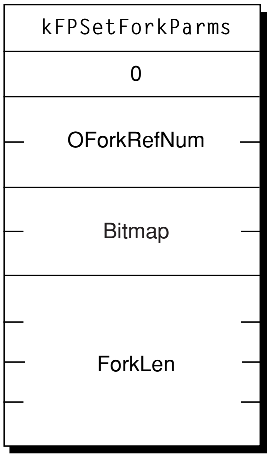
Table 3-60 Result codes for the FPSetForkParms command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.
kFPBitmapErr	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
kFPDiskFull	No more space exists on the volume.
kFPLockErr	Range lock conflict exists.
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or fork reference number is invalid.
kFPVolLocked	Volume is ReadOnly.

Figure 3-69 shows the request block for the FPSetForkParms command.

Figure 3-69 Request block for the FPSetForkParms command

Request



FPSetVolParms

Sets a volume's backup date.

```
byte CommandCode
byte Pad
short VolumeID
short Bitmap
Date BackupDate
```

Parameter Descriptions

`CommandCode`

kFPSetVolParms (32).

`Pad`

Pad byte.

`VolumeID`

Volume ID.

`Bitmap`

Bitmap describing the parameters to be set. This parameter is the same as the `Bitmap` parameter for the `FPGetVolParms` (page 183) command, but only the Backup Date bit can be set. For bit definitions for this bitmap, see "Volume Bitmap" (page 257).

`BackupDate`

New backup date.

`Result`

`kFPNoErr` if no error occurred. See Table 3-61 for the possible result codes.

`ReplyBlock`

None.

Discussion

The user must have previously called FPOpenVol (page 209) for this volume.

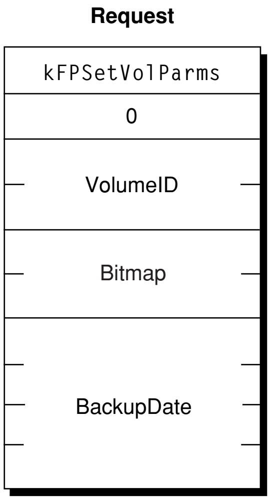
Table 3-61 lists the result codes for the FPSetVolParms command.

Table 3-61 Result codes for the FPSetVolParms command

Result code	Explanation
kFPAccessDenied	User does not have the access privileges required to use this command.
kFPBitmapErr	Attempt was made to set a parameter that cannot be set by this command; bitmap is null.
kFPMiscErr	Non-AFP error occurred.
kFPParamErr	Session reference number or Volume ID is unknown.
kFPVolLocked	Volume is ReadOnly.

Figure 3-70 shows the request block for the FPSetVolParms command.

Figure 3-70 Request block for the FPSetVolParms command



FPWrite

Writes a block of data to an open fork.

```
byte CommandCode
byte Flag
short OForkRefNum
long Offset
long ReqCount
ForkData
```

Parameter Descriptions

`CommandCode`

`kFPWrite` (33).

`Flag`

Bit 7 is the `StartEndFlag` bit, and it indicates whether `Offset` is relative to the beginning or end of the fork. A value of zero indicates that the start is relative to the beginning of the fork; a value of 1 indicates that the start is relative to the end of the fork.

`OForkRefNum`

Open fork reference number.

`Offset`

Byte offset from the beginning or the end of the fork indicating where the write is to begin; a negative value indicates a byte within the fork relative to the end of the fork.

`ReqCount`

Number of bytes to be written.

`ForkData`

Data to be written, which is not a part of the request block. Instead, the data is transmitted to the server in an intermediate exchange of DSI packets.

`Result`

`kFPNoErr` if no error occurred. See Table 3-62 (page 245) for the possible result codes.

`ActualCount`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block consisting of a long, called `LastWritten`, containing the number of the byte just past the last byte written.

Discussion

The server writes data to the open fork, starting at the number of bytes from the beginning or end of the fork as specified by `Offset`. The `StartEndFlag` bit indicates whether the block of data is to be written at an offset relative to the beginning or the end of the fork. When the offset is relative to the end of the fork, data can be written without knowing the exact end of the fork, which is useful when multiple writers modify a fork concurrently. The server returns the number of the byte just past the last byte written.

This command differs from the `FPWriteExt` (page 246) command in that the `FPWriteExt` command is prepared to handle the large values that may be required for writing to files that reside in volumes larger than 4 GB in size.

If the block of data to be written extends beyond the end of the fork, the fork is extended. If part of the range is locked by another user, the server returns a `kFPLockErr` result code and does not write any data to the fork.

The file's Modification Date is not changed until the fork is closed.

The fork must be open for writing by the user sending this command.

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

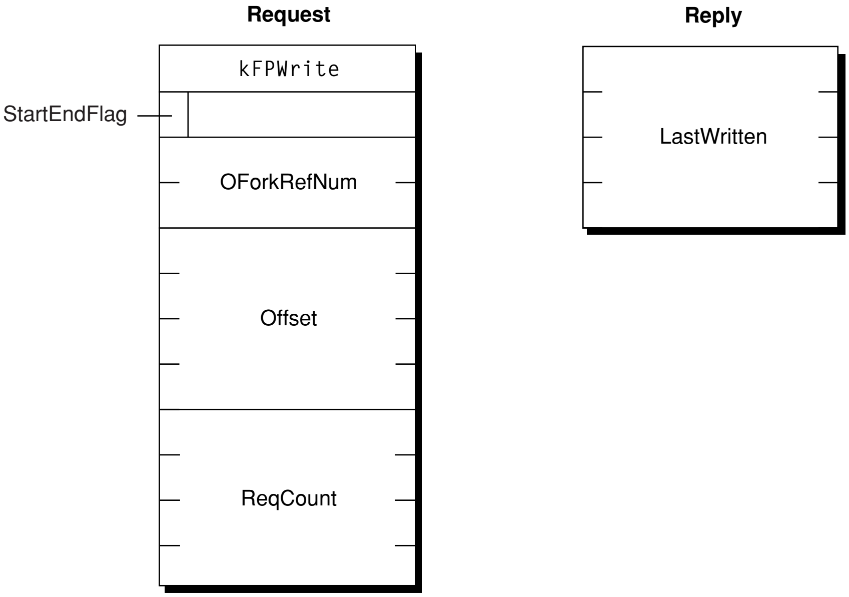
Table 3-62 lists the result codes for the `FPWrite` command.

Table 3-62 Result codes for the `FPWrite` command

Result code	Explanation
`kFPAccessDenied`	Fork is not open for writing by this user.
`kFPDiskFull`	No space exists on the volume.
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number or open fork reference number is unknown.

Figure 3-71 shows the request and reply blocks for the `FPWrite` command.

Figure 3-71 Request and reply blocks for the `FPWrite` command



FPWriteExt

Writes a block of data to an open fork.

```
byte CommandCode
byte Flag
short OForkRefNum
long long Offset
long long ReqCount
ForkData
```

Parameter Descriptions

`CommandCode`

`kFPWriteExt` (61).

`Flag`

Bit 7 of the `Flag` parameter is the `StartEndFlag` bit, and it indicates whether `Offset` is relative to the beginning or end of the fork. A value of zero indicates that the start is relative to the beginning of the fork; a value of 1 indicates that the start is relative to the end of the fork.

`OForkRefNum`

Open fork reference number.

`Offset`

Byte offset from the beginning or the end of the fork indicating where the write is to begin; a negative value indicates a byte within the fork relative to the end of the fork.

`ReqCount`

Number of bytes to be written.

`ForkData`

Data to be written, which is not a part of the request block. Instead, the data is transmitted to the server in an intermediate exchange of DSI packets.

`Result`

`kFPNoErr` if no error occurred. See Table 3-63 (page 248)for the possible result codes.

`ActualCount`

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

`ReplyBlock`

If the result code is `kFPNoErr`, the server returns a reply block consisting of a long, called `LastWritten`, containing the number of the byte just past the last byte written.

Discussion

The server writes data to the open fork, starting at the number of bytes from the beginning or end of the fork as specified by `Offset`.

This command differs from the `FPWrite` (page 243) command in that this command is prepared to handle the large values that may be required for writing to files that reside in volumes larger than 4 GB in size.

The `StartEndFlag` bit indicates whether the block of data is to be written at an offset relative to the beginning or the end of the fork. When the offset is relative to the end of the fork, data can be written without knowing the exact end of the fork, which is useful when multiple writers modify a fork concurrently. The server returns the number of the byte just past the last byte written.

If the block of data to be written extends beyond the end of the fork, the fork is extended. If part of the range is locked by another user, the server returns a `kFPLockErr` result code and does not write any data to the fork.

The file's Modification Date is not changed until the fork is closed.

The fork must be open for writing by the user sending this command.

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

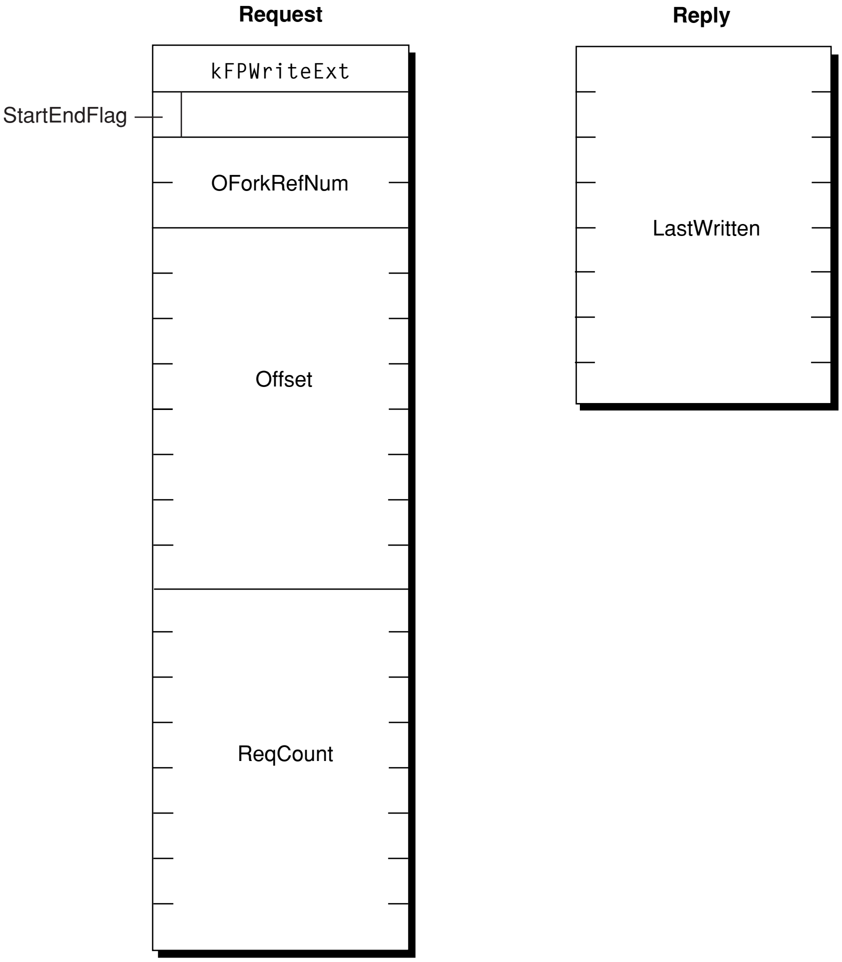
Table 3-63 lists the result codes for the `FPWriteExt` command.

Table 3-63 Result codes for the `FPWriteExt` command

Result code	Explanation
`kFPAccessDenied`	Fork is not open for writing by this user.
`kFPDiskFull`	No space exists on the volume.
`kFPLockErr`	Some or all of the requested range is locked by another user.
`kFPMiscErr`	Non-AFP error occurred.
`kFPParamErr`	Session reference number or open fork reference number is unknown.

Figure 3-72 shows the request and reply blocks for the `FPWriteExt` command.

Figure 3-72 Request and reply blocks for the `FPWriteExt` command



FPZzzzz

Notifies the server that the client is going to sleep.

```
byte CommandCode
byte Pad
unsigned long Flags
```

Parameter Descriptions

`CommandCode`

kFPZzzzz (122).

`Flag`

Reserved.

`ReplyBlock`

None.

Discussion

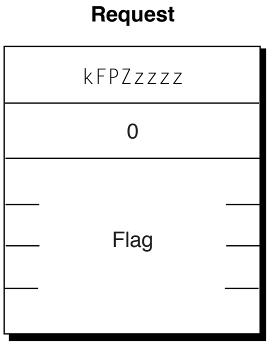
If an AFP sharepoint is mounted when the client goes to sleep (for example, an idle sleep or a demand sleep such as when the lid of a PowerBook is closed), the client sends the FPZzzzz command. This command notifies the AFP server that the client is going to sleep and that the server should not send any more packets to the client. When the client is awakens, it will send AFP packets to the server, which notifies the server that the client is now awake.

The AFP server should have a setting for the maximum time that a client can sleep — typically 24 hours. If a client has been asleep longer than the maximum sleep time, the server assumes that the client has been disconnected and may free client-related resources on the server.

The FPZzzzz command is supported by AFP 2.3 and later over AFP/TCP only.

Figure 3-73 shows the request block for the FPZzzzz command.

Figure 3-73 Request and reply blocks for the FPWriteExt command



Apple Filing Protocol Data Types

Access Rights Bitmap

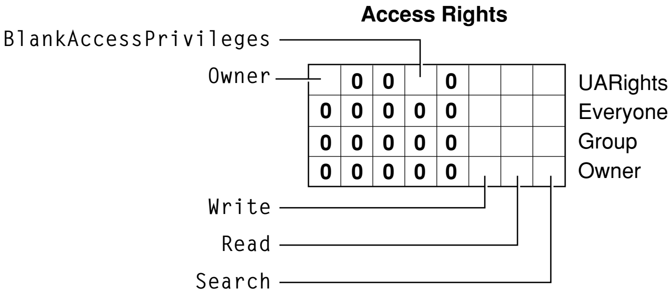
A 32-bit value whose bits indicate the ability of the directory's Owner, Group, and Everyone to read, write, and search the directory.

Discussion

Call FPGetFileDirParms (page 156) to get the Access Rights bitmap.

Figure 3-74 (page 251) shows the Access Rights bitmap.

Figure 3-74 Access Rights bitmap



Directory Bitmap

A 16-bit value whose bits are used to get and set directory parameters.

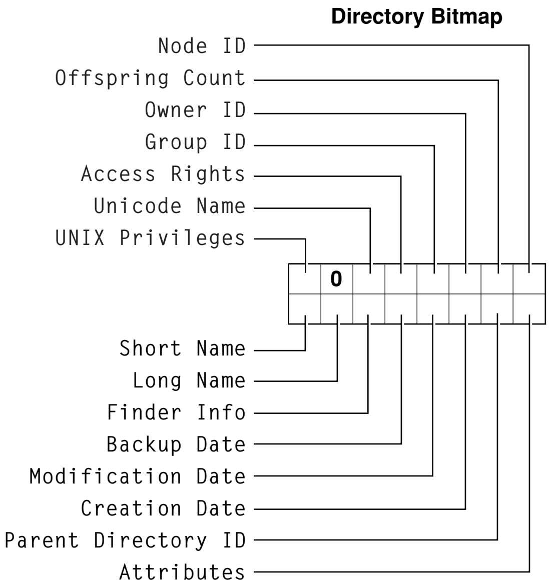
```
enum {
    kFPAttributeBit = 0x1,
    kFPParentDirIDBit = 0x2,
    kFPCreateDateBit = 0x4,
    kFPModDateBit = 0x8,
    kFPBackupDateBit = 0x10,
    kFPFinderInfoBit = 0x20,
    kFPLongNameBit = 0x40,
    kFPShortNameBit = 0x80,
    kFPNodeIDBit = 0x100,
    kFPOffspringCountBit = 0x0200,
    kFPOwnerIDBit = 0x0400,
    kFPGroupIDBit = 0x0800,
    kFPAccessRightsBit = 0x1000,
    kFPUTF8NameBit = 0x2000,
    kFPUnixPrivsBit = 0x8000
};
```

Discussion

The Directory bitmap is used when calling FPGetFileDirParms (page 156) to indicate the directory parameters you want to get. It is also used when calling FPSetDirParms (page 230) and FPSetFileDirParms (page 233) to set directory parameters.

Figure 3-75 (page 252) describes the Directory bitmap.

Figure 3-75 Directory bitmap



Directory Attributes Bitmap

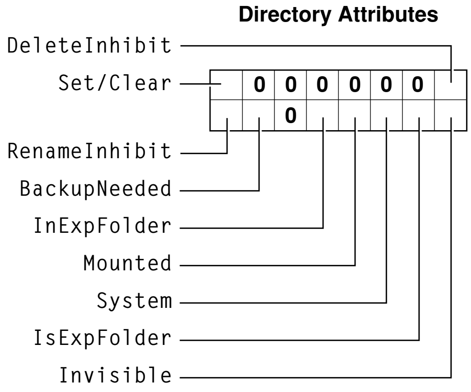
A 16-bit value whose bits provide additional information about a directory.

Discussion

Use the bits in the Directory Attributes bitmap to inhibit renaming or deleting the directory. Other bits in the Directory Attributes bitmap indicate whether the directory needs to be backed up, whether the directory is mounted by a user, whether the directory is invisible or a system directory, and whether the directory is in a shared area or is a share point. When calling FPSetDirParms (page 230) and FPSetFileDirParms (page 233) to set Directory Attributes, use the Set/Clear bit (bit 15) to indicate whether you are setting or clearing a directory attribute.

Figure 3-76 (page 253) describes the Attributes bitmap for a directory.

Figure 3-76 Directory Attributes bitmap



File Bitmap

A 16-bit value whose bits are used to get and set file parameters.

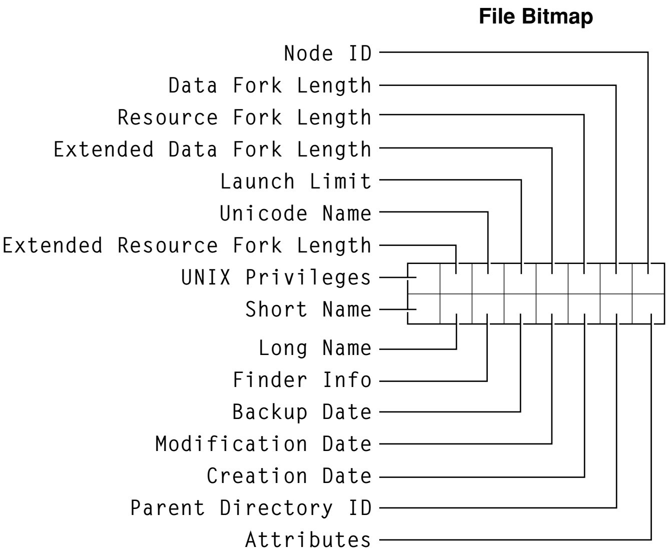
```
enum {
    kFPAttributeBit = 0x1,
    kFPParentDirIDBit = 0x2,
    kFPCreateDateBit = 0x4,
    kFPModDateBit = 0x8,
    kFPBackupDateBit = 0x10,
    kFPFinderInfoBit = 0x20,
    kFPLongNameBit = 0x40,
    kFPShortNameBit = 0x80,
    kFPNodeIDBit = 0x100,
    kFPDataForkLenBit = 0x0200,
    kFPRsrcForkLenBit = 0x0400,
    kFPExtDataForkLenBit = 0x0800,
    kFPLaunchLimitBit = 0x1000,
    kFPUTF8NameBit = 0x2000,
    kFPExtRsrcForkLenBit = 0x4000
    kFPUnixPrivsBit = 0x8000
};
```

Discussion

The File bitmap is used when calling FPGetFileDirParms (page 156) to indicate the file parameters you want to get. It is also used when calling FPSetFileParms (page 236) and FPSetFileDirParms (page 233) to set file parameters.

Figure 3-77 (page 254) describes the File bitmap.

Figure 3-77 File bitmap



File Attributes Bitmap

A 16-bit value whose bits provide additional information about a file.

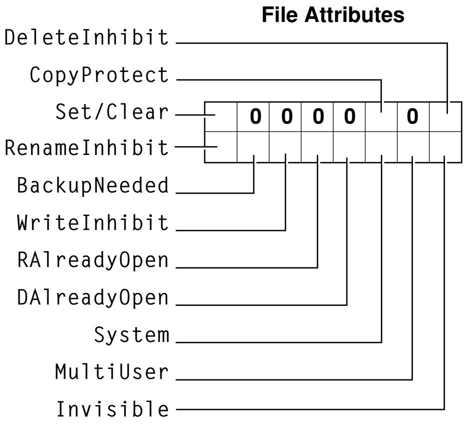
```
enum {
    kFPInvisibleBit = 0x01,
    kFPMultiUserBit = 0x02,
    kFPSystemBit = 0x04,
    kFPDAlreadyOpenBit = 0x08,
    kFPRAlreadyOpenBit = 0x10,
    kFPWriteInhibitBit = 0x20,
    kFPBackUpNeededBit = 0x40,
    kFPRenameInhibitBit = 0x80,
    kFPDeleteInhibitBit = 0x100,
    kFPCopyProtectBit = 0x400,
    kFPSetClearBit = 0x8000
};
```

Discussion

Use the bits in the File Attributes bitmap to inhibit writing, renaming or deleting the file. Other bits in the File Attributes bitmap indicate whether the file needs to be backed up, whether the file can be copied, whether the file is invisible or a system file, whether the file's data or resource fork is open, and whether the file can be opened at the same time by multiple users. When calling `FPSetFileParms` (page 236) and `FPSetFileDirParms` (page 233) to set File Attributes, use the Set/Clear bit (bit 15) to indicate whether you are setting or clearing a file attribute.

Figure 3-78 (page 255) describes the Attributes bitmap for a file.

Figure 3-78 File Attributes bitmap



FPUnixPrivs

A structure the describes UNIX privileges for files and directories that reside on a volume that supports UNIX privileges.

```
struct FPUnixPrivs {
    unsigned long uid;
    unsigned long gid;
    unsigned long permissions;
    unsigned long ua_permissions;
};
```

Field Descriptions

`uid`

User ID of the file or directory's owner.

`gid`

Group ID of the file or directory's owner.

`permissions`

Setting of the file or directory's permission bits.

`ua_permissions`

User's access rights to the file or directory. Bit 31 is set if the user is the owner of the file or directory.

Discussion

A FPUnixPrivs structure is returned when you call FPGetFileDirParms (page 156) and specify that you want to get the UNIX privileges for a file or directory.

Server Flags Bitmap

A 16-bit value that describes server capabilities.

```
enum {
    kSupportsCopyfile = 0x01,
    kSupportsChgPwd = 0x02,
    kDontAllowSavePwd = 0x04,
    kSupportsSrvrMsg = 0x08,
    kSrvrSig = 0x10,
    kSupportsTCP = 0x20,
    kSupportsSrvrNotify = 0x40,
    kSupportsReconnect = 0x80,
    kSupportsDirServices = 0x100,
    kSupportsUTF8Name = 0x200
    kSupportsSuperClient = 0x8000
};
```

Discussion

The Server Flags bitmap is returned by the FPGetSrvrInfo (page 169) command.

Volume Attributes Bitmap

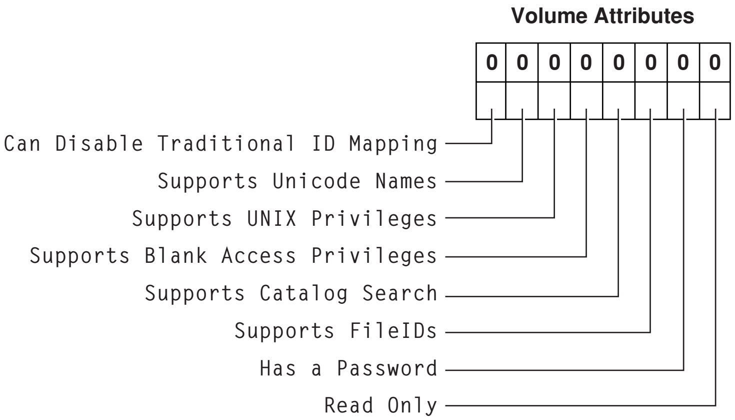
A 16-bit value whose bits describe how a volume is mounted and whether it supports certain AFP features.

```
enum {
    kReadOnly = 0x01,
    kHasVolumePassword = 0x02,
    kSupportsFileIDs = 0x04,
    kSupportsCatSearch = 0x08,
    kSupportsBlankAccessPrivs = 0x10,
    kSupportsUnixPrivs = 0x20,
    kSupportsUTF8Names = 0x40,
    kNoNetworkUserIDs = 0x80
};
```

Discussion

Figure 3-79 describes the Attributes bitmap for a volume.

Figure 3-79 Volume Attributes bitmap



Volume Bitmap

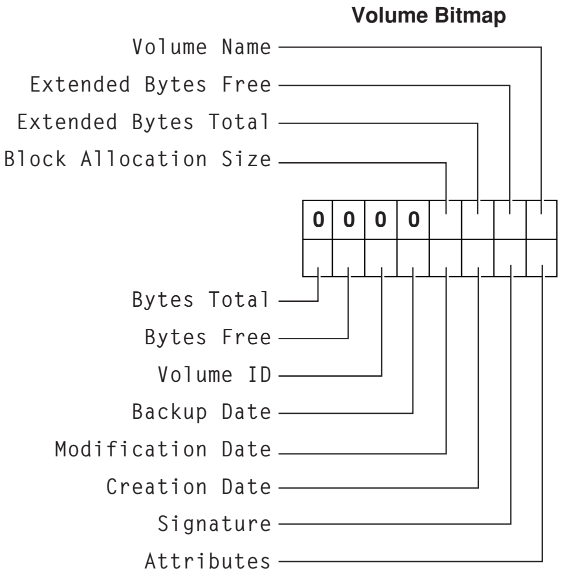
A 16-bit value whose bits are used to get and set volume parameters.

```
enum {
    kFPVolAttributeBit = 0x1,
    kFPVolSignatureBit = 0x2,
    kFPVolCreateDateBit = 0x4,
    kFPVolModDateBit = 0x8,
    kFPVolBackupDateBit = 0x10,
    kFPVolIDBit = 0x20,
    kFPVolBytesFreeBit = 0x40,
    kFPVolBytesTotalBit = 0x80,
    kFPVolNameBit = 0x100,
    kFPVolExtBytesFreeBit = 0x200,
    kFPVolExtBytesTotalBit = 0x400,
    kFPVolBlockSizeBit = 0x800
};
```

Discussion

The Volume bitmap is used when calling FPGetVolParms (page 183) to indicate the volume parameters you want to get. It is also used when calling FPSetVolParms (page 242) to set a volume's backup date, which is the only Volume parameter that an AFP client can set. Figure 3-80 (page 258) describes the Volume bitmap.

Figure 3-80 Volume bitmap



Apple Filing Protocol Constants

AFP Version Strings

Strings that identify different AFP versions.

```
"AFPVersion 2.1"
"AFP2.2"
"AFP2.3"
"AFPX02"
"AFP3.1"
```

Constant Descriptions

AFPVersion 2.1

AFP version 2.1.

AFP2.2

AFP version 2.2.

AFP2.3

AFP version 2.3.

`AFPX02`

AFP version 3.0.

`AFPX31`

AFP version 3.1.

Discussion

AFP Version strings are returned by the `FPGetSrvrInfo` (page 169) command. AFP clients sent an AFP Version string as a parameter to the `FPLogin` (page 185) and `FPLoginExt` (page 191) commands.

AFP UAM Strings

Strings that identify different UAM versions.

```
"No User Authent"
"Cleartxt Passwrd"
"Randum Exchange"
"2-Way Randnum"
"DHCAST128"
"DHX2"
"Client Krb v2"
“Recon1”
```

Constant Descriptions

`No User Authent`

UAM that does not require user authentication.

`Cleartxt Passwrd`

Cleartext Password UAM.

`Randum Exchange`

Randum Number Exchange UAM.

`2-Way Randnum`

Two-Way Randum Number Exchange UAM.

`DHCAST128`

Diffie-Hellman Exchange UAM to be used during the log process.

`DHX2`

Causes the Diffie-Hellman Exchange 2 UAM.

`Client Krb v2`

Kerberos UAM.

`Recon1`

Reconnect UAM.

Discussion

AFP UAM strings are returned by the `FPGetSrvrInfo` (page 169) command. AFP clients sent an AFP UAM string as a parameter to the `FPLogin` (page 185) and `FPLoginExt` (page 191) commands.

FPGetSessionToken Types

Values for the Type parameter of the FPGetSessionToken command.

```
enum {
kLoginWithoutID = 0,
kLoginWithID = 1,
kReconnWithID = 2,
kLoginWithTimeAndID = 3,
kReconnWithTimeAndID = 4,
kRecon1Login = 5,
kRecon1ReconnectLogin = 6,
kRecon1Refresh = 7,kGetKerberosSessionKey = 8
};
```

Constant Descriptions

kLoginWithoutID

The FPGetSessionToken parameter block does not contain an ID parameter; specified by AFP clients that support a version of AFP prior to AFP 3.1.

kLoginWithID

Deprecated.

kReconnWithID

Deprecated.

kLoginWithTimeAndID

The FPGetSessionToken parameter block contains an ID and a time stamp parameter. The command is sent to indicate that the client wants its old session to be discarded.

kReconnWithTimeAndID

The FPGetSessionToken parameter block contains an ID and a time stamp parameter. The command is sent to indicate that the client has successfully reconnected and wants the session to be updated with the new value of ID.

kRecon1Login

Used after logging in to get a credential that can be used to reconnect using the Reconnect UAM. Specifying kRecon1Login tells the server to destroy any old sessions that may be associated with the ID parameter to the FPGetSessionToken command.

kRecon1ReconnectLogin

Used to get a new reconnect token after reconnecting using the Reconnect UAM.

kRecon1RefreshToken

Used to get a new credential when the current credential is about to expire.

kGetKerberosSessionKey

Used to get a Kerberos v5 session key.

Discussion

The value of the Type parameter determines the behavior of the server when it receives theFPGetSessionToken (page 166) command.

Apple Filing Protocol Result Codes

The result codes specific to the Apple Filing Protocol are listed in Table 3-64.

Table 3-64 Apple Filing Protocol result codes

Constant	Value	Description
`kFPAccessDenied`	–5000	User does not have the access privileges required to use the command.
`kFPAuthContinue`	–5001	Authentication is not yet complete.
`kFPBadUAM`	–5002	Specified UAM is unknown
`kFPBadVersNum`	–5003	Server does not support the specified AFP version.
`kFPBitmapErr`	–5004	Attempt was made to get or set a parameter that cannot be obtained or set with this command, or a required bitmap is null
`kFPCantMove`	–5005	Attempt was made to move a directory into one of its descendent directories.
`kFPDenyConflict`	–5006	Specified fork cannot be opened because of a deny modes conflict.
`kFPDirNotEmpty`	–5007	Directory is not empty.
`kFPDiskFull`	–5008	No more space exists on the volume
`kFPEOFErr`	–5009	No more matches or end of fork reached.
`kFPFileBusy`	–5010	When attempting a hard create, the file already exists and is open.
`kFPFlatVol`	–5011	Volume is flat and does not support directories.
`kFPItemNotFound`	–5012	Specified APPL mapping, comment, or icon was not found

		in the Desktop database; specified ID is unknown.

Constant	Value	Description
kFPLockErr	–5013	Some or all of the requested range is locked by another user; a lock range conflict exists.
kFPMiscErr	–5014	Non-AFP error occurred.
kFPNoMoreLocks	–5015	Server's maximum lock count has been reached.
kFPNoServer	–5016	Server is not responding.
kFPObjectExists	–5017	File or directory already exists.
kFPObjectNotFound	–5018	Input parameters do not point to an existing directory, file, or volume.
kFPParamErr	–5019	Session reference number, Desktop database reference number, open fork reference number, Volume ID, Directory ID, File ID, Group ID, or subfunction is unknown; byte range starts before byte zero; pathname is invalid; pathname type is unknown; user name is null, exceeds the UAM's user name length limit, or does not exist, MaxReplySize is too small to hold a single offspring structure, ThisUser bit is not set, authentication failed for an undisclosed reason, specified user is unknown or the account has been disabled due to too many login attempts; ReqCount or Offset is negative; NewLineMask is invalid.
kFPRangeNotLocked	–5020	Attempt to unlock a range that is locked by another user or that is not locked at all.
kFPRangeOverlap	–5021	User tried to lock some or all of a range that the user has already locked.
kFPSessClosed	–5022	Session is closed.
kFPUserNotAuth	–5023	UAM failed (the specified old password doesn't match); no

Constant	Value	Description
		user is logged in yet for the specified session; authentication failed; password is incorrect.
`kFPCallNotSupported`	–5024	Server does not support this command.
`kFPObjectTypeErr`	–5025	Input parameters point to the wrong type of object.
`kFPTooManyFilesOpen`	–5026	Server cannot open another fork.
`kFPServerGoingDown`	–5027	Server is shutting down.
`kFPCantRename`	–5028	Attempt was made to rename a volume or root directory.
`kFPDirNotFound`	–5029	Input parameters do not point to an existing directory.
`kFPIconTypeError`	–5030	New icon's size is different from the size of the existing icon.
`kFPVolLocked`	–5031	Volume is Read Only.
`kFPObjectLocked`	–5032	File or directory is marked DeleteInhibit; directory being moved, renamed, or moved and renamed is marked RenameInhibit; file being moved and renamed is marked RenameInhibit; attempt was made to open a file for writing that is marked WriteInhibit; attempt was made to rename a file or directory that is marked RenameInhibit.
`kFPContainsSharedErr`	–5033	Directory contains a share point.
`kFPIDNotFound`	–5034	File ID was not found. (No file thread exists.)
`kFPIDExists`	–5035	File already has a File ID.
`kFPDiffVolErr`	–5036	Wrong volume.
`kFPCatalogChanged`	–5037	Catalog has changed.
`kFPSameObjectErr`	–5038	Two objects that should be different are the same object.
`kFPBadIDErr`	–5039	File ID is not valid.

Constant	Value	Description
`kFPPwdSameErr`	–5040	User attempted to change his or her password to the same password that is currently set.
`kFPPwdTooShortErr`	–5041	User password is shorter than the server's minimum password length, or user attempted to change password to a password that is shorter than the server's minimum password length.
`kFPPwdExpiredErr`	–5042	User's password has expired.
`kFPInsideSharedErr`	–5043	Directory being moved contains a share point and is being moved into a directory that is shared or is the descendent of a directory that is shared.
`kFPInsideTrashErr`	–5044	Shared directory is being moved into the Trash; a directory is being moved to the trash and it contains a shared folder.
`kFPPwdNeedsChangeErr`	–5045	User's password needs to be changed.
`kFPPwdPolicyErr`	–5046	New password does not conform to the server's password policy.

Document Revision History

This table describes revisions to *Apple Filing Protocol*.

Date	Notes
Oct. 16, 2003	Updated to cover Apple Filing Protocol features as of version 3.1.