Mac OS X Server
Developer's Kit

Apple Filing Protocol Client

January 2001

Contents

Figures and Tables 5
Conventions Used in This Manual 7
For More Information 8

Chapter 1 Introduction to the Apple Filing Protocol Client 9

Data Stream Interface 10
Implementation 10
The DSI Header 11
Data Stream Commands 13
Getting Status from the DSI 13
Opening a Session with the DSI 14
Sending AFP Commands 15
Writing Data 16
Detecting Timeouts and Disconnections 16
Receiving Attention Information 16
Closing a DSI Session 21

Chapter 2 Apple Filing Protocol Client Reference 23

Using the AFP Client Library 23
AFPLibraryPresent 24
AFPLibraryVersion 24
AFPCreateSharedVolumesEnumerator 25
AFPCreateSVEFromAddress 28
AFPGetSharedVolumesCount 29
AFPGetIndexedSharedVolume 30
AFPSortSharedVolumes 31
AFPMountSharedVolume 32
AFPMountSharedVolumeOnMP 33
AFPGetLoginInformation 34
AFPGetMountAtStartup 35

AFPSetMountAtStartup 36
AFPChangePassword 37
AFPDeleteSharedVolumesEnumerator 37
AFP Client Application-Defined Routines 38
Notification Callback Routine 39
Filter Callback Routine 40
System Event Callback Routine 40
Using the Data Stream Interface 41
DSGetStatus 42
DSOpenSession 43
DSCommand 44
DSWrite 46
DSCloseSession 47
Using AFP URLs 48
NewAFPURL 49
AFPMountURL 50
IsAFPURL 51
ParseAFPURL 52
DisposeAFPURL 53
Result Codes 54

Appendix A AFP and PAP URL Formats 55

Apple Filing Protocol URLs 55
AppleTalk AFP URL Format 56
TCP/IP AFP URL Format 56
Printer Access Protocol URLs 56
AppleTalk PAP URL Format 56
TCP/IP PAP URL Format 57

Appendix B Mount Flags 59

Index 61

Figures and Tables

Chapter 1 **Introduction to the Apple Filing Protocol Client** 9

Figure 1-1 AFP client architecture 9
Figure 1-2 DSI header format 11
Figure 1-3 Format of options in the DSOpenSession packet 14
Figure 1-4 Format of the AFPUserBytes field 17
Figure 1-5 Attention code bits in AFPUserBytes 17
Figure 1-6 Format of long attention message 21

Table 1-1 Fields in the DSI header 12
Table 1-2 Data stream commands 13
Table 1-3 Option fields in the DSOpenSession packet 15
Table 1-4 Attention code bits 18
Table 1-5 Attention code bit combinations 19

About This Manual

This manual describes the application programming interface for the AFP client software, which consists of functions for creating and disposing of shared volume enumerator references and mounting shared volumes, functions for sending Data Stream Interface (DSI) commands, and functions for creating, parsing, and disposing of AFP Uiversal Resource Locators (URLs).

Conventions Used in This Manual

The Courier font is used to indicate server control calls, code, and text that you type. Terms that are defined in the glossary appear in boldface at first mention in the text. This guide include 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

Inside Mac OS X provides important information for all developers of software for Mac OS X Server

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

- *Inside Mac OS X: Directory Services.* Apple Computer, Inc.

For information on the programming interface for developing a Directory Services plug-in, see

- *Inside Mac OS X: Directory Services Plug-ins.* Apple Computer, Inc.

For information on version 3.0 of the Apple Filing Protocol, see

- *Mac OS X Server Developer's Kit: Apple Filing Protocol Version 3.0*, Apple Computer, Inc.

For information on administering and using Mac OS X Server, see

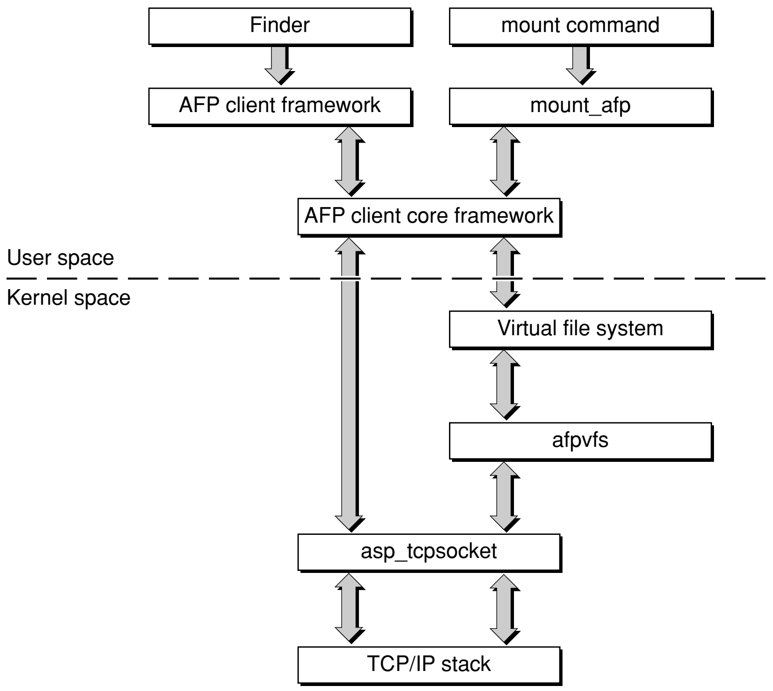
- *Mac OS X Server Administrator's Guide*, Apple Computer, Inc.
- *Getting Started with Mac OS X Server*, Apple Computer, Inc.

Introduction to the Apple Filing Protocol Client

This manual describes the programming interface for developing Apple Filing Protocol (AFP) client software for Mac OS X. The programming interface provides functions for creating and managing AFP Universal Resource Locators (URLs) and for creating and managing shared volume enumerators.

Figure 1-1 illustrates the architecture for the AFP client software in Mac OS X.

Figure 1-1 AFP client architecture



Note
The functions described in this manual should not be called at deferred task time. ◆

Data Stream Interface

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

Note
AFP 3.0 does not run over AppleTalk. AFP 3.0 supports pathnames in Unicode encoding, which could result in command and reply blocks that are too long to fit in AppleTalk packets. ◆

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

Implementation

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

- It registers the AFP server on a well-known data stream port. For TCP, the port number is 548. Protocol suites that include a service-locating protocol can be used to advertise and locate an AFP server. For example, NBP can be used for AFP over ADSP, and the Service Advertisement Protocol (SAP) can be used for AFP over IPX/SPX.

- It uses a request/response model that supports multiple outstanding requests on any given connection. In other words, the request's window size may be greater than 1 in length.
- It replies to multiple outstanding requests in any order.
- It provides a one-to-one mapping between the AFP session and the port ID or connection ID maintained by the data stream protocol.
- It maintains some state information for every open AFP client connection. This allows the server to demultiplex requests to an appropriate AFP session.
- It allows the AFP server to send and receive large packets. The size of the packets is based on the underlying network's maximum transmission unit (MTU).

The DSI Header

The DSI prepends the header shown in Figure 1-2 to every AFP request and reply.

Figure 1-2 DSI header format

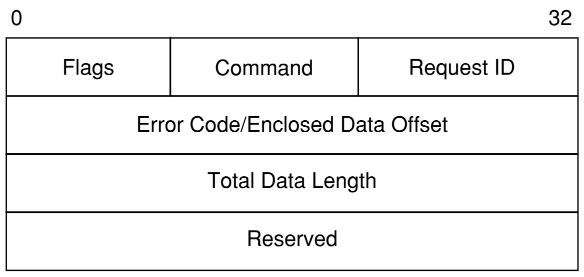


Table 1-1 describes each field in the DSI header.

Table 1-1 Fields in the DSI header

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

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

Error Code/ Enclosed Data Offset	In request packets, this field is ignored by the server for all commands except `DSWrite`. For future compatibility, AFP clients should set this field to zero for all commands except `DSWrite`. In request packets for which the command is `DSWrite`, this field contains a data offset that is the number of bytes in the data representing AFP command information. The server uses this information to collect the AFP command part of the packet before it accepts the data that corresponds to the packet. For example, when an AFP client sends an `FPWrite` command to write data on the server, the enclosed data offset would be 12. In reply packets, this field contains an error code.

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

Field	Purpose
Total Data Length	A 32-bit unsigned value that specifies the total length of the data that follows the data stream header.
Reserved	A 32-bit field reserved for future use. AFP clients should set this field to zero.

Data Stream Commands

DSI commands are similar to ASP commands, and they preserve all of the ASP commands except `ASPWriteContinue`. The DSI commands are listed in Table 1-2.

Table 1-2 Data stream commands

Command code	Command name	Originator of command
1	`DSCloseSession`	AFP client or AFP server
2	`DSCommand`	AFP client
3	`DSGetStatus`	AFP client
4	`DSOpenSession`	AFP client
5	`DSTickle`	AFP client or AFP server
6	`DSWrite`	AFP client
8	`DSAttention`	AFP server

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

Getting Status from the DSI

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

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

Opening a Session with the DSI

The `DSOpenSession` command is usually the first command that the AFP client sends once it has established a connection with an AFP server. (Alternatively, the AFP client may first send a `DSGetStatus` command. In this case, the AFP server immediately tears down the connection after delivering the requested status information.) The `DSOpenSession` command opens a DSI session and delivers the AFP client's first AFP command, which must be `FPLogin`, `FPLoginExt`, or `FPGetAuthMethods`.

The data portion of a `DSOpenSession` packet may contain options defined by the AFP client (request) or AFP server (reply). The options must conform to the format shown in Figure 1-3.

Figure 1-3 Format of options in the DSOpenSession packet

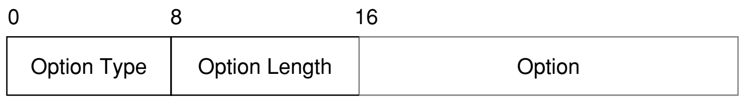


Table 1-3 describes each field in the option portion of the `DSOpenSession` packet.

Table 1-3 Option fields in the DSOpenSession packet

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

Sending AFP Commands

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

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

Writing Data

The `DSWrite` command sends an `FPAddIcon`, `FPWrite`, or an `FPWriteExt` command and associated data to an AFP server. The amount of data to be written may be up to the size of the server request quantum described earlier in the section "Opening a Session with the DSI" (page 14).

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

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

Detecting Timeouts and Disconnections

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

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

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

Receiving Attention Information

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

The *AFPUserBytes* field makes up the two-byte attention code sent in an DSI Attention packet to the AFP client. This section describes how the *AFPUserBytes* field supports the server message and auto-reconnect features.

The format of the *AFPUserBytes* field is shown in Figure 1-4.

Figure 1-4 Format of the AFPUserBytes field

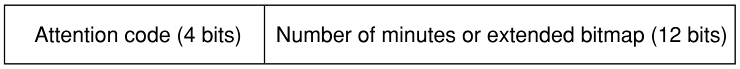
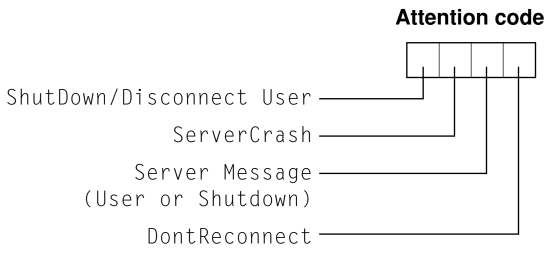


Figure 1-5 shows how the attention code bits in the *AFPUserBytes* field are defined.

Figure 1-5 Attention code bits in AFPUserBytes



The bit numbers for the attention code bits are listed in Table 1-4.

Table 1-4 Attention code bits

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

Table 1-5 lists valid combinations of the attention code bits.

Table 1-5 Attention code bit combinations

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

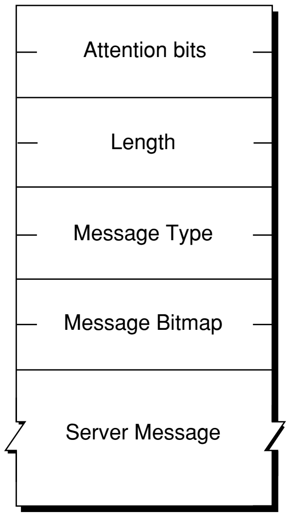
Table 1-5 Attention code bit combinations (continued)

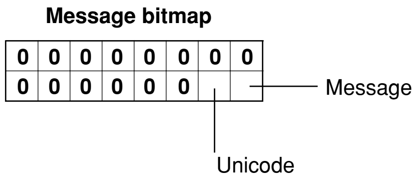
Combination	Meaning
0010	The server has a server message available for this AFP client. The AFP client should immediately submit an `FPGetSrvrMsg` command to receive and display the message. The extended bitmap is reserved for Apple Computer's use only.
0011	Server Notification. The server is notifying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

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

An AFP 3.0 client can inform the server that it can accept long attention messages (for example, .5K to 3K) by setting the attention quantum size in the *Option* field of the `DSOpenSession` command. (For details, see Table 1-3.) If the server chooses, it may then include the attention message in the initial attention data instead of in a later packet. The format of a long attention message is shown in Figure 1-6.

Figure 1-6 Format of long attention message





Closing a DSI Session

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

Note
Using `DSCommand` to send the `FPLogout` command does not close the DSI session. ◆

Apple Filing Protocol Client Reference

This chapter describes the AFP Client Library functions, the Data Stream Interface functions, and the AFP URL functions.

Using the AFP Client Library

This section describes the functions that make up the AFP Client Library. The header file for these functions is `afpClient.h`, located in `/System/Library/Frameworks/AppleShareClientLibraryCore.framework/Headers`. The functions are

- `AFPLibraryPresent` (page 24), which determines whether the AFP Client Library is available.
- `AFPLibraryVersion` (page 24), which obtains the version of the AFP Client Library.
- `AFPCreateSharedVolumesEnumerator` (page 25), which creates a shared volumes enumerator reference from a server name and a server zone.
- `AFPCreateSVEFromAddress` (page 28), which creates a shared volumes enumerator reference from an address stored in a `sockaddr` structure.
- `AFPGetIndexedSharedVolume` (page 30), which obtains the name of a shared volume by its index number.
- `AFPSortSharedVolumes` (page 31), which sorts the list of volumes in a shared volume enumerator reference.
- `AFPMountSharedVolume` (page 32), which mounts a shared volume.
- `AFPMountSharedVolumeOnMP` (page 33), which mounts a shared volume.

- `AFPGetLoginInformation` (page 34), which obtains the log on type (Guest or registered user). If the user is a registered user, `AFPGetLoginInformation` also obtains the user's name and password.
- `AFPGetMountAtStartup` (page 35), which obtains the startup mounting state of a shared volume.
- `AFPSetMountAtStartup` (page 36), which sets the startup mounting state of a shared volume.
- `AFPChangePassword` (page 37), which sets the startup mounting state of a shared volume.
- `AFPDeleteSharedVolumesEnumerator` (page 37), which disposes of a shared volume enumerator reference created by `AFPCreateSharedVolumesEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28).

AFPLibraryPresent

Determines whether the AFP Client Library is available.

```
Boolean AFPLibraryPresent (void);
```

function result The `AFPLibraryVersion` function returns `TRUE` if the AFP Client library is available and `FALSE` if the library is not available.

DISCUSSION

The `AFPLibraryPresent` function determines whether the AFP Client library is available.

AFPLibraryVersion

Obtains the version of the AFP Client library.

```
UInt32 AFPLibraryVersion (void);
```

function result The `AFPLibraryVersion` function returns an unsigned 32-bit value containing the version number.

DISCUSSION

The `AFPLibraryVersion` function obtains the version number of the AFP Client library.

AFPCreateSharedVolumesEnumerator

Uses a server name and zone to create a shared volumes enumerator reference.

```
OSStatus AFPCreateSharedVolumesEnumerator (StringPtr serverName,
                     StringPtr serverZone,
                     StringPtr uamName,
                     StringPtr userName,
                     StringPtr password,
                     AShareEventUPP callback,
                     void * evtContext,
                     ATFilterUPP filter,
                     void * filterParam,
                     ATNotifyUPP notifier,
                     void * contextPtr;
                     AFPSharedVolumesEnumeratorRef * Ref);
```

`serverName` On input, a value of type `StringPtr` that points to a string containing the name of the AFP server for which the shared volume enumerator reference is being created. For TCP/IP connections, `serverName` should be the DNS name of the server.

`serverZone` On input, a value of type `StringPtr` that points to a string containing the name of the AppleTalk zone in which the server specified by `serverName` resides. For TCP/IP connections, set `serverZone` to `NULL`. If the user logs on to a server using AppleTalk as the transport protocol, the enumerator reference created by `AFPCreateSharedVolumesEnumerator` is updated with the name of the zone in which `serverName` resides.

`uamName`	On input, a value of type `StringPtr` that points to a string containing the name of the UAM to use when authenticating the user identified by the `userName` parameter, or `NULL`. If `uamName` is `NULL` and if the user provides a name in the log on dialog box, that is displayed by calling `AFPGetSharedVolumesCount` (page 29) the enumerator reference is updated with the name of the UAM that authenticated the user.
`userName`	On input, a value of type `StringPtr` that points a string containing the name of the user to authenticate, or `NULL`. If `userName` is `NULL` and if the user enters a name in the log on dialog box that is displayed by calling `AFPGetSharedVolumesCount` (page 29), the enumerator reference is updated with the name that the user entered.
`password`	On input, a value of type `StringPtr` that points to a string containing the password that is to be used to authenticate the user name specified by the `userName` parameter, or `NULL`. If `password` is `NULL` and if the user enters a password in the log on dialog box that is displayed by calling `AFPGetSharedVolumesCount` (page 29), the enumerator reference is updated with the password that the user entered.
`callback`	On input, a value of type `AShareEventUPP` that points to an application-defined system event callback routine (page 40) that handles events that occur while `AFPGetSharedVolumesCount` (page 29) or other AFP Client library functions display dialog boxes, or `NULL`. If `callback` is `NULL`, the calling application will not receive update events while these dialog boxes are displayed.
`evtContext`	On input, an untyped pointer to arbitrary data that `AFPCreateSharedVolumesEnumerator` passes to the application-defined system event callback routine specified by `callback`, or `NULL`. Your application can use `evtContext` to associate the invocation of your system event callback routine with any particular enumerator reference.
`filter`	On input, a value of type `ATFilterUPP` that points an optional application-defined filter routine (page 40) that can be used to control the volumes that are included in the enumerator reference, or `NULL` to match all volumes.

`filterparam`	On input, an untyped pointer to arbitrary data that `AFPCreateSharedVolumesEnumerator` passes to the filter routine specified by `filter`, or `NULL`. Your application can use `filterparam` to associate the invocation of your filter routine with any particular enumerator reference.
`notifier`	On input, a value of type `ATNotifyUPP` that points to an application-defined notification routine (page 39) that is to be called when address resolution for `serverName` is complete, or `NULL` if your application does not provide a notification routine.
`contextPtr`	On input, an untyped pointer to arbitrary data that `AFPCreateSharedVolumesEnumerator` passes to the notification routine specified by the `notifier` parameter. Your application can use `contextPtr` to associate the invocation of your notification routine with any particular enumerator reference.
`ref`	On input, a value of type `AFPSharedVolumesEnumerator`. On output, `ref` points to the enumerator reference created by `AFPCreateSharedVolumesEnumerator`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPCreateSharedVolumesEnumerator` function creates a shared volumes enumerator reference that can be passed as a parameter to `AFPGetSharedVolumesCount`, `AFPGetIndexedSharedVolume`, `AFPSortSharedVolumes`, and `AFPMountSharedVolumes` or `AFPMountSharedVolumesOnMP` if the enumerator reference was created for a TCP/IP connection.

Passing the enumerator reference to `AFPGetSharedVolumesCount` (page 29) obtains the number of volumes that the user has permission to mount.

Passing the enumerator reference and an index number to `AFPGetIndexedSharedVolume` (page 30) obtains the name of the volume that is associated with the specified index number.

Passing the enumerator reference to `AFPSortSharedVolumes` (page 31) returns a sorted list of volume names. If your application needs to allow the user to select one or more volumes for mounting, it can display the sorted list in a dialog box.

Passing the enumerator reference and the name of a volume to be mounted to `AFPMountSharedVolume` (page 32) causes the specified volume to be mounted.

Passing the enumerator reference, the name of a volume to be mounted, a password for the volume, and mount flags to `AFPMountSharedVolumeOnMP` (page 33) causes the specified volume to be mounted.

When you no longer need the enumerator reference, call `AFPDeleteSharedVolumesEnumerator` (page 34) to deallocate the memory that has been allocated to it.

AFPCreateSVEFromAddress

Uses a `sockaddr` structure to create a shared volumes enumerator reference.

```
OSStatus AFPCreateSVEFromAddress(AddressPtr serverAddress,
                        StringPtr uamName,
                        StringPtr userName,
                        StringPtr password,
                        AFPSharedVolumesEnumeratorRef * ref);
```

`serverAddress`	On input, a value of type `AddressPtr` that points to a `sockaddr` structure containing the address of the server for which the shared volumes enumerator is to be created.
`uamName`	On input, a value of type `StringPtr` that points to a string containing the name of the UAM to use when authenticating the user identified by the `userName` parameter, or `NULL` if authentication is not required.
`userName`	On input, a value of type `StringPtr` that points a string containing the name of the user to authenticate, or `NULL` if a user name is not required.
`password`	On input, a value of type `StringPtr` that points to a string containing the password that is to be used to authenticate the user name specified by the `userName` parameter, or `NULL` if a password is not required.
`ref`	On input, a pointer to a value of type `AFPSharedVolumesEnumeratorRef`. On output, `ref` contains a shared volumes enumerator that pass to `AFPMountSharedVolume` (page 32).

function result A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPCreateSVEFromAddress` function creates a shared volumes enumerator for the server identified by the `serverAddress` parameter that can be passed as a parameter to `AFPGetSharedVolumesCount`, `AFPGetIndexedSharedVolume`, `AFPSortSharedVolumes`, and `AFPMountSharedVolumes` or `AFPMountSharedVolumesOnMP`.

Passing the enumerator reference to `AFPGetSharedVolumesCount` (page 29) obtains the number of volumes that the user has permission to mount.

Passing the enumerator reference and an index number to `AFPGetIndexedSharedVolume` (page 30) obtains the name of the volume that is associated with the specified index number.

Passing the enumerator reference to `AFPSortSharedVolumes` (page 31) returns a sorted list of volume names. If your application needs to allow the user to select one or more volumes for mounting, it can display the sorted list in a dialog box.

Passing the enumerator reference to `AFPMountSharedVolume` (page 32) and the name of a volume causes the specified volume to be mounted.

Passing the enumerator reference, the name of a volume to be mounted, a password for the volume, and mount flags to `AFPMountSharedVolumeOnMP` (page 33) causes the specified volume to be mounted.

When you no longer need the enumerator reference, call `AFPDeleteSharedVolumesEnumerator` (page 34) to deallocate the memory that has been allocated to it.

AFPGetSharedVolumesCount

Obtains the number of shared volumes that the user has permission to mount.

```
OSStatus AFPGetSharedVolumesCount (
                        AFPSharedVolumesEnumeratorRef ref,
                        Boolean * allfound,
                        UInt32 * count);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28) that represents an AFP server.
`allfound`	On input, a pointer to a Boolean value. On output, `allfound` points to a value that is `TRUE` if all volumes have been counted and that is `FALSE` if `AFPGetSharedVolumesCount` is still counting. If `allfound` is `FALSE`, call `AFPGetSharedVolumesCount` again until `allfound` is `TRUE`.
`count`	On input, a pointer to an unsigned 32-bit integer. On output, `count` points to a value that contains the current count of the number of volumes the user has permission to mount.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPGetSharedVolumesCount` function returns the number of volumes that a the user has permission to mount. Once an application obtains the number of volumes that the user has permission to mount, it can call `AFPGetIndexedSharedVolume` (page 30) to obtain the name of each volume by its index number.

If the shared volume enumerator reference specified by `ref` does not contain a user name or password, `AFPGetSharedVolumesCount` causes a log on dialog box to be displayed. The log on dialog box allows the user to log in as Guest or as a registered user with an optional password. After the user enters this information, the enumerator reference is updated with log on type (Guest or registered user) and the name and password (if any) the user entered.

AFPGetIndexedSharedVolume

Obtains the name of a shared volume by its index number.

```
OSStatus AFPGetIndexedSharedVolume (AFPSharedVolumesEnumeratorRef ref,
                        OneBasedIndex index,
                        StringPtr volumeName);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28) that represents the AFP server that shares the volume whose name is to be obtained.
`index`	On input, a value of type `OneBasedIndex` that specifies the index number. Call `AFPGetSharedVolumesCount` (page 29) to determine the highest valid value of `index`. The lowest value of `index` is 1.
`volumeName`	On input, a value of type `StringPtr`. On output, `volumeName` points to the name of the volume that corresponds to the specified index value.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPGetIndexedSharedVolume` function obtains the name of a volume by its index number. To determine the highest possible index number, call `AFPGetSharedVolumesCount` (page 30).

Once you obtain the name of a volume, you can sort the list of volume names by calling `AFPSortSharedVolumes` (page 31) and you can mount a particular volume by calling `AFPMountSharedVolume` (page 32) or `AFPMountSharedVolumeOnMP` (page 33) if the enumerator reference was created for a TCP/IP connection.

AFPSortSharedVolumes

Sorts the names of shared volumes.

```
OSStatus AFPSortSharedVolumes (AFPSharedVolumesEnumeratorRef ref);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28).
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPSortSharedVolumes` function sorts the list of volumes in a shared volume enumerator reference using the binary presentation of the characters that make up each volume name in ascending order.

AFPMountSharedVolume

Mounts a shared volume.

```
OSStatus AFPMountSharedVolume (AFPSharedVolumesEnumeratorRef ref,
                        Str255 volumeName,
                        short * volumeRefNum,
                        Boolean * isMounted);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPMountSharedVolumeOnMP` (page 33) if the enumerator reference was created for a TCP/IP connection.
`volumeName`	On input, a value of type `Str255` that specifies the name of the volume that is to be mounted. Call `AFPGetIndexedSharedVolume` (page 30) to obtain the volume name.
`volumeRefNum`	On input, a pointer to a value of type `short`. On output, `volumeRefNum` points to a unique volume reference number that your application can use to refer to the volume when it sends AFP commands to the server.
`isMounted`	On input, a pointer to a Boolean whose value is `TRUE` if your application wants `AFPMountSharedVolumes` to return an error if the volume is already mounted. Set `isMounted` to `NULL` if you don't want `AFPMountSharedVolumes` to return an error if the volume is already mounted.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPMountSharedVolumes` function mounts the specified shared volume. If a volume is already mounted and if the `isMounted` parameter is `TRUE`, `AFPMountSharedVolumes` returns an error.

AFPMountSharedVolumeOnMP

Mounts a shared volume.

```
AFPMountSharedVolumeOnMP(AFPSharedVolumesEnumeratorRef  ref,
                     StringPtr inVolumeName,
                     const UInt8* inVolPassword,
                     const char* inMountPoint,
                     UInt32 inMountFlags,
                     UInt32 inAltFlags,
                     Boolean inMakeUnique,
                     UInt32 inMaxPath,
                     char* outMountPath);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSVEFromAddress` (page 28).
`inVolumeName`	On input, a value of type `StringPtr` that points to a string containing the name of the volume that is to be mounted. Call `AFPGetIndexedSharedVolume` (page 30) to obtain the volume name.
`inVolPassword`	On input, a pointer to a value of type `UInt8` containing the volume's password.
`inMountPoint`	On input, a pointer to a character string containing the mount point path.
`inMountFlags`	On input, a value of type `UInt32` containing mount flags. The mount flags are the same flags that are used for the `mount`(2) system call. For a list of possible values, see Appendix B.

`inAltFlags`	On input, a value of type `UInt32` containing alternate mount flags. The alternate mount flags are the same flags that are used for the `mount`(2) system call. For a list of possible values, see Appendix B.
`inMakeUnique`	On input, a pointer to a Boolean value. If `inMakeUnique` is `TRUE` and another file system is already mounted on the path pointed to by `inMountPoint`, the volume is mounted using a algorithm that creates a unique mount point path name. If `inMakeUnique` is `FALSE`, `AFPMountSharedVolumeOnMP` fails of a file system is already mounted on the path pointed to by `inMountPoint`.
`inMaxPath`	On input, a value of type `UInt32` that specifies the maximum length of `outMountPath`.
`outMountPath`	On input, a pointer to a character string containing the mount path for the mounted volume.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPMountSharedVolumeOnMP` function mounts the shared volume represented by `ref`.

AFPGetLoginInformation

Obtains login information from a shared volume enumerator reference.

```
OSStatus AFPGetLoginInformation (
                        AFPSharedVolumesEnumeratorRef ref,
                        Boolean * isGuest,
                        Str255 userName,
                        Str255 password);
```

`ref`	On input, a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28) that has been used to log a user into an AFP server.

`isGuest`	On input, a pointer to a Boolean value. On output, `isGuest` points to a value that is `TRUE` if the user chose to log on as Guest or `FALSE` if the user chose to log on as a registered user.
`userName`	On input, a value of type `Str255`. On output, `userName` contains the name the user typed in the Name text box of the log on dialog box displayed by `AFPGetSharedVolumesCount`.
`password`	On input, a value of type `Str255`. On output, `password` contains the password (if any) the user typed in the Password text box of the log on dialog box displayed by `AFPGetSharedVolumesCount`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPGetLoginInformation` obtains log on information from an enumerator reference that has been used to log a user on to an AFP server.

AFPGetMountAtStartup

Obtains a volume's startup mount information.

```
OSStatus AFPGetMountAtStartup (
                        AFPSharedVolumesEnumeratorRef * ref,
                        StringPtr volumeName);
```

`ref`	On input, a pointer to a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28).
`volumeName`	On input, a value of type `StringPtr` that points to the name of the volume.
function result	A result code whose value is `noErr` if the volume is set to be mounted at startup and whose value is `nsvErr` if the volume is not set to be mounted at startup.

DISCUSSION

The `AFPGetMountAtStartup` function obtains the startup mount information for a volume as set in the specified shared volume enumerator reference.

AFPSetMountAtStartup

Sets a volume's startup mount information.

```
OSStatus AFPSetMountAtStartup (
                    AFPSharedVolumesEnumeratorRef * ref,
                    StringPtr volumeName,
                    Boolean toMount);
```

`ref`	On input, a pointer to a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28) that identifies the volume that is to be set.
`volumeName`	On input, a value of type `StringPtr` that points to the name of the volume whose startup mount information is to be set.
`toMount`	On input, a Boolean whose value is `TRUE` if the volume is to be mounted when the computer starts up or `FALSE` if the volume is not to be mounted at startup.
function result	A result code whose value is `noErr` if the volume's startup mounting information was successfully set. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPSetMountAtStartup` function updates the specified shared volume enumerator reference with the latest startup mount information for a volume.

AFPChangePassword

Changes the specified password.

```
AFPChangePassword (AFPSharedVolumesEnumeratorRef * ref,
                    StringPtr oldPassword,
                    StringPtr newPassword);
```

`ref`	On input, a pointer to a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25) or `AFPCreateSVEFromAddress` (page 28).
`oldPassword`	On input, a value of type `StringPtr` that points to a string containing the password that is to be changed.
`newPassword`	On input, a value of type `StringPtr` that points to a string containing the password that is to be set.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPChangePassword` function changes the specified password.

AFPDeleteSharedVolumesEnumerator

Disposes of a shared volume enumerator reference.

```
OSStatus AFPDeleteSharedVolumesEnumerator (
                    AFPSharedVolumesEnumeratorRef * ref);
```

`ref`	On input, a pointer to a value of type `AFPSharedVolumesEnumeratorRef` created by previously calling `AFPCreateSharedVolumeEnumerator` (page 25).
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `AFPDeleteSharedVolumesEnumerator` function disposes of a shared volume enumerator reference and deallocates memory that has been allocated for it. You should dispose of the enumerator reference as soon as it has fulfilled its purpose of mounting shared volumes.

The enumerator reference maintains an open session with the AFP server that is separate from sessions for any of the AFP server's volumes that have been mounted. Disposing of the enumerator reference closes this session.

Note
The `AFPDeleteSharedVolumesEnumerator` function deallocates memory, so your application should call it during main event time. ◆

AFP Client Application-Defined Routines

This section describes three application-defined routines that your application can provide when it calls `AFPCreateSharedVolumesEnumerator` (page 25):

- A notification callback routine that is called when the host name specified as a parameter to `AFPCreateSharedVolumesEnumerator` is resolved into an IP address.
- A filter callback routine that is called to control the display of volume names.
- A system event callback routine that is called when update events occur while an AFP Client API function displays a dialog box.

Notification Callback Routine

Your notification callback routine is called when the host name specified as an parameter to `AFPCreateSharedVolumesEnumerator` (page 25) is resolved into an IP address. This is how you would declare your notification callback routine if you were to name it `MyURLNotifyUPP`:

```
OSStatus MyURLNotifyUPP (void* userContext,
                         ATEventCode code,
                         OSStatus result,
                         void *cookie);
```

`userContext`	An application-defined value that your application previously passed as the `contextPtr` parameter when it called `AFPCreateSharedVolumesEnumerator` (page 25).
`code`	A value of type `ATEventCode` specifying the event that triggered the callback. The value of `code` is `AT_SHAREDVOLUMES_COMPLETE`.
`result`	A value of type `OSStatus`. A value of `noErr` indicates that the `AFPCreateSharedVolumesEnumber` successfully created a shared volume enumerator reference.
`cookie`	An untyped pointer to arbitrary data. For details about `cookie`, see *Inside Macintosh: Networking with Open Transport*.
result	Your notification callback routine should always return `noErr`.

DISCUSSION

Your notification callback routine should use the `userContext` parameter to determine which enumerator reference has been successfully created. Your application can then pass the enumerator reference to `AFPGetSharedVolumesCount` (page 29) to determine the number of volumes that the server identified by the enumerator reference shares.

Filter Callback Routine

Your filter callback routine is called to control the display of volume names. This is how you would declare your filter callback routine if you were to name it `MyFilterUPP`.

```
void MyFilterUPP(StringPtr name,
                          void *data);
```

`name`	A value of type `StringPtr` that points to a volume name.
`data`	An untyped pointer to arbitrary data that your application passed as the `filterParam` parameter when it called `AFPCreateSharedVolumesEnumerator` (page 25).
result	Your filter callback routine should return `TRUE` to display the volume name and `FALSE` to prevent the volume name from being displayed.

DISCUSSION

Your filter callback routine should determine whether the volume identified by `name` should be displayed.

System Event Callback Routine

Your system event callback routine is called to handle update events that may occur while `AFPGetSharedVolumesCount` (page 29) displays the log on dialog box. Here is how you would declare your system event callback routine if you were to name it `MyAShareEventUPP`.

```
void MyAShareEventUPP(EventRecord *theEvent,
                          void *contextPtr);
```

`theEvent`	A pointer to an event record that describes the event that occurred.

`event`	An untyped pointer to arbitrary data that your application passed as the `evtContext` parameter when it called `AFPCreateSharedVolumesEnumerator` (page 25). For more information on the `EventRecord` structure see *Inside Macintosh: Overview.*
result	Your system event callback routine should process the system event and return `noErr`.

DISCUSSION

Your system event callback routine may be called to handle events that occur while `AFPGetSharedVolumesCount` (page 29) displays the log on dialog box. Your system event callback routine should process the event and return `noErr`.

Using the Data Stream Interface

This section describes the functions that make up the Data Stream Interface (DSI). The header file for these functions is `afpDatastream.h`, located in `/System/Library/Frameworks/AppleShareClientLibraryCore.framework/Headers`. The functions are

- `DSGetStatus` (page 42), which gets status information from an AFP server without opening a DSI session.
- `DSOpenSession` (page 43), which opens a DSI session with an AFP server.
- `DSCommand` (page 44), which sends an AFP command to an AFP server.
- `DSWrite` (page 46), which writes data to an AFP server.
- `DSCloseSession` (page 47), which closes a DSI session.

DSGetStatus

Gets status.

```
OSStatus DSGetStatus(struct sockaddr* inConnectAddr,
                     char *inReplyBuffer,
                     UInt32 inReplyBufferLen,
                     UInt32* outAFPCmdResult,
                     UInt32* outRcvdReplyLen);
```

`inConnectAddr`	On input, a pointer to a value of type `sockaddr` that specifies the IP address of the server with which the connection is to be established.
`inReplyBuffer`	On input, a pointer to a buffer in which the AFP server is to place the requested status information. The buffer must be more than 2048 bytes in length.
`inReplyBufferLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the buffer pointed to by `inReplyBuffer`.
`outAFPCmdResult`	On input, a pointer to a value of type `UInt32`. On output, `outAFPCmdResult` points to the result code for this command.
`outRcvdReplyLen`	On input, a pointer to a value of type `UInt32`. On output, `outRcvdReplyLen` points to the length of the status information returned by the AFP server in the buffer pointed to by `inReplyBuffer`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `DSGetStatus` function gets status information from an AFP server.

To get status information, the AFP client must establish a connection on the server's listening port. The AFP client then sends a `DSGetStatus` command to the server. After delivering the requested status information, the AFP server immediately tears down the connection.

DSOpenSession

Opens a DSI session.

```
OSStatus DSOpenSession(struct sockaddr* inConnectAddr,
                    char* inCommand,
                    UInt32 inCommandLen,
                    char* inReplyBuffer,
                    UInt32 inReplyBufferLen,
                    UInt32* outAFPCmdResult,
                    UInt32* outRcvdReplyLen,
                    AFPSocketDesc* outSD,
                    AFPSocketID* outSockID);
```

`inConnectAddr`	On input, a pointer to a value of type `sockaddr` that specifies the IP address of the server with which the connection is to be established.
`inCommand`	On input, a pointer to a null-terminated string containing an `FPLogin` or `FPLoginExt` command. For information on these and other AFP commands, see *Apple Filing Protocol Version 3.0*.
`inCommandLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the command specified by `inCommand`.
`inReplyBuffer`	On input, a pointer to a buffer in which the AFP server is to place the reply block for the `AFPLogin` or `AFPLoginExt` command.
`inReplyBufferLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the buffer pointed to by `inReplyBuffer`.
`outAFPCmdResult`	On input, a pointer to a value of type `UInt32`. On output, `outAFPCmdResult` points to the result code for the command specified by `inCommand`.
`outRcvdReplyLen`	On input, a pointer to a value of type `UInt32`. On output, `outRcvdReplyLen` points to the length of the reply block returned by the AFP server in the buffer pointed to by `inReplyBuffer`.

`outSD`	On input, a pointer to a value of type `AFPSocketDesc`. On output, `outSD` points the socket descriptor that the AFP server assigned for this attempt to open an AFP session.
`outSockID`	On input, a pointer to a value of type `AFPSocketID`. On output, `outSockID` points the socket ID that the AFP server assigned for this attempt to open an AFP session.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `DSOpenSession` function opens a session with the DSI and can be used to send one of three AFP commands to an AFP server: `FPLogin`, `FPLoginExt`, and `FPGetAuthMethods`.

The data portion of a `DSOpenSession` packet may contain options defined by the AFP client (request) or AFP server (reply). For details, see "Opening a Session with the DSI" (page 14).

DSCommand

Sends a command to an AFP server.

```
OSStatus DSCommand(AFPSocketDesc inSD,
                     AFPSocketID inSockID,
                     char* inCommand,
                     UInt32 inCommandLen,
                     char* inReplyBuffer,
                     UInt32 inReplyBufferLen,
                     UInt32* outAFPCmdResult,
                     UInt32* outRcvdReplyLen);
```

`inSD`	On input, a value of type `AFPSocketDesc` returned by a previous `DSOpenSession` call.
`inSockID`	On input, a value of type `AFPSocketID` returned by a previous `DSOpenSession` call.

`inCommand`	On input, a pointer to a null-terminated string containing an AFP command. For information on these and other AFP commands, see *Apple Filing Protocol Version 3.0*.
`inCommandLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the command specified by `inCommand`.
`inReplyBuffer`	On input, a pointer to a buffer in which the AFP server is to place the reply block for the AFP command specified pointed to by `inCommand`.
`inReplyBufferLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the buffer pointed to by `inReplyBuffer`.
`outAFPCmdResult`	On input, a pointer to a value of type `UInt32`. On output, `outAFPCmdResult` points to the result code for the command specified by `inCommand`.
`outRcvdReplyLen`	On input, a pointer to a value of type `UInt32`. On output, `outRcvdReplyLen` points to the length of the reply block returned by the AFP server in the buffer pointed to by `inReplyBuffer`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `DSCommand` function sends AFP commands other than `FPAddIcon`, `FPGetAuthMethods`, `FPLogin`, `FPLoginExt`, `FPLogout`, `FPWrite`, and `FPWriteExt`. For more information about how the DSI interacts with an AFP server when the DSI processes an AFP command sent by `DSCommand`, see "Sending AFP Commands" (page 15).

DSWrite

Sends a write command or an add icon command to an AFP server.

```
OSStatus DSWrite(AFPSocketDesc inSD,
                     AFPSocketID inSockID,
                     char* inCommand,
                     UInt32 inCommandLen,
                     char* inReplyBuffer,
                     UInt32 inReplyBufferLen,
                     char* inWriteBuffer,
                     UInt32 inWriteBufferLen,
                     UInt32* outAFPCmdResult,
                     UInt32* outRcvdReplyLen);
```

`inSD`	On input, a value of type `AFPSocketDesc` returned by a previous `DSOpenSession` call.
`inSockID`	On input, a value of type `AFPSocketID` returned by a previous `DSOpenSession` call.
`inCommand`	On input, a pointer to a null-terminated string containing an `FPWrite`, `FPWriteExt`, or `FPAddIcon` command. For information on these and other AFP commands, see *Apple Filing Protocol Version 3.0*.
`inCommandLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the command specified by `inCommand`.
`inReplyBuffer`	On input, a pointer to a buffer in which the AFP server is to place the reply block for the AFP command specified pointed to by `inCommand`.
`inWriteBuffer`	On input, a pointer to a string containing the data that is to be written.
`inReplyBufferLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the buffer pointed to by `inWriteBuffer`.
`outAFPCmdResult`	On input, a pointer to a value of type `UInt32`. On output, `outAFPCmdResult` points to the result code for the command specified by `inCommand`.

`outRcvdReplyLen`	On input, a pointer to a value of type `UInt32`. On output, `outRcvdReplyLen` points to the number of the byte just past the last byte written.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `DSWrite` function sends an `FPAddIcon`, `FPWrite`, or `FPWriteExt` command and the command's associated data to an AFP server. For more information about how the DSI interacts with an AFP server when the DSI processes an AFP command sent by `DSWrite`, see "Writing Data" (page 16).

DSCloseSession

Closes a DSI session.

```
OSStatus DSCloseSession(AFPSocketDesc inSD,
                        AFPSocketID inSockID,
                        char* inCommand,
                        UInt32 inCommandLen,
                        UInt32* outAFPCmdResult);
```

`inSD`	On input, a value of type `AFPSocketDesc` returned by a previous `DSOpenSession` call.
`inSockID`	On input, a value of type `AFPSocketID` returned by a previous `DSOpenSession` call.
`inCommand`	On input, a pointer to a null-terminated string containing an `FPLogout` command. For information on these and other AFP commands, see *Apple Filing Protocol Version 3.0.*
`inCommandLen`	On input, a pointer to a value of type `UInt32` that specifies the length of the command specified by `inCommand`.

`outAFPCmdResult`
: On input, a pointer to a value of type `UInt32`. On output, `outAFPCmdResult` points to the result code for the command specified by `inCommand`.

function result A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `DSCloseSession` function closes a DSI session. The AFP client can immediately tear down the DSI session and reclaim all of the resources it has allocated to it.

Note
Calling `DSCommand` to send an `FPLogout` command does not close the AFP session. ◆

Using AFP URLs

This section describes functions that create, mount, verify, parse and dispose of AFP URLs. The functions are:

- `NewAFPURL` (page 49), which creates an AFP URL.
- `AFPMountURL` (page 50), which mounts an AFP URL.
- `IsAFPURL` (page 51), which determines whether a character string is a valid AFP URL.
- `ParseAFPURL` (page 52), which parses an AFP URL into its component parts.
- `DisposeAFPURL` (page 53), which disposes of an AFP URL.

The header file for the functions described in this section is `afpURL.h`, located in `/System/Library/Frameworks/AppleShareClientLibraryCore.framework/Headers`.

NewAFPURL

Creates an AFP URL.

```
char * NewAFPURL (StringPtr protocolName,
                  StringPtr serverNameOrHost
                  StringPtr zoneNameOrNULL,
                  StringPtr uamName
                  StringPtr userName
                  StringPtr password,
                  StringPtr volume,
                  StringPtr path);
```

`protocolName`	On input, a value of type `StringPtr` that points to a string containing the transport protocol. Specify `at` for AppleTalk or `ip` for TCP/IP. If `protocolName` is `NULL`, TCP/IP is assumed.
`serverNameOrHost`	On input, a value of type `StringPtr` that points to a string containing the name or address of the computer that hosts the URL that is being created. The name can be a Network Bind Protocol (NBP) name, a Domain Name System (DNS) name, or an Internet Protocol (IP) address.
`zoneNameOrNull`	On input, a value of type `StringPtr` that points to a string containing the AppleTalk zone in which the computer that hosts the URL resides, or `NULL` if the computer does not reside in an AppleTalk zone.
`uamName`	On input, a value of type `StringPtr` that points to a string `containing the name of the user authentication module (UAM)` that is to be used to authenticate the user specified by the `userName` parameter.
`userName`	On input, a value of type `StringPtr` that points to a string containing the user name that is to be authenticated.
`password`	On input, a value of type `StringPtr` that points to a string containing the password that is to be used to authenticate the user specified by the `userName` parameter.

`volume`	On input, a value of type `StringPtr` that points to a string containing the volume that is to be mounted if authentication is successful.
`path`	On input, a value of type `StringPtr` that points to a string containing the pathname for a particular directory or file on the volume specified by the `volume` parameter. The path should use the forward slash character (`/`) to delimit the directory and filename components of the path.
function result	The `AFPNewURL` function returns a pointer to the character string that contains the new AFP URL.

DISCUSSION

The `NewAFPURL` function creates an AFP URL that contains all of the information needed to authenticate a user on a particular server, including the transport protocol, the server name, the zone name (if the transport protocol is AppleTalk), the user authentication module that is to be used to authenticate the user, the user name and his or her password, and the volume that is to be mounted.

AFPMountURL

Mounts an AFP URL.

```
OSStatus AFPMountURL(const char* inURL,
                     const char* inMountPoint,
                     UInt32 inMountFlags,
                     UInt32 inAltFlags);
```

`inURL`	On input, a pointer to a character string containing the URL to mount. The value pointed to by `inURL` must be a fully qualified URL, including the user name and password, and (optionally) the user authentication method to use.
`inMountPoint`	On input, a pointer to a character string containing the mount point path.

`inMountFlags`	On input, a value of type `UInt32` containing mount flags. The mount flags are the same flags that are used for the `mount`(2) system call. For a list of possible values, see Appendix B.
`inAltFlags`	On input, a value of type `UInt32` alternate mount flags. The mount flags are the same flags that are used for the `mount`(2) system call. For a list of possible values, see Appendix B.
function result	A result code. For a list of possible result codes, see “Result Codes” (page 54).

DISCUSSION

The `AFPMountURL` function mounts the URL pointed to by `inURL` on the mount point pointed to by `inMountPoint`.

Note
Calling the `AFPMountURL` function does not cause any dialog boxes to be displayed. ◆

IsAFPURL

Verifies an AFP URL.

```
Boolean IsAFPURL (char * url);
```

`url`	On input, a pointer to a character string that contains an AFP URL previously created by calling `NewAFPURL` (page 49).
function result	The `IsAFPURL` function returns `TRUE` if the `url` parameter points to an AFP URL and `FALSE` if it does not.

DISCUSSION

The `ISAFPURL` function verifies that a character string is a properly formatted AFP URL.

ParseAFPURL

Parses an AFP URL.

```
OSStatus ParseAFPURL (char * url,
                      StringPtr protocolName,
                      StringPtr serverNameOrHost
                      StringPtr zoneNameOr,
                      StringPtr uamName
                      StringPtr userName
                      StringPtr password,
                      StringPtr volume,
                      StringPtr path);
```

`url`	On input, a pointer to a character string containing an AFP URL previously created by calling `NewAFPURL` (page 49).
`protocolName`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `protocolName` contains the protocol name obtained from the AFP URL specified by the `url` parameter, or is `NULL` if `url` was not created with a protocol name.
`serverNameOrHost`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `serverNameOrHost` contains the server or host name obtained from the AFP URL specified by the `url` parameter, or is `NULL` if `url` was not created with a server or host name.
`zoneNameOr`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `zoneNameOr` contains the zone name obtained from the AFP URL specified by the `url` parameter, or is `NULL` if `url` was not created with a zone name.
`uamName`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `uamName` contains the UAM name obtained from the AFP URL specified by the `url` parameter, or `NULL` if `url` was not created with the name of a UAM.

`userName`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `userName` contains the user name obtained from the AFP URL specified by the `url` parameter, or `NULL` if `url` was not created with a user name.
`password`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `password` contains the password obtained from the AFP URL specified by the `url` parameter, or `NULL` if `url` was not created with a password.
`volume`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `volume` contains the volume name obtained from the AFP URL specified by the `url` parameter, or `NULL` if `url` was not created with a volume name.
`path`	On input, a value of type `StringPtr` that points to a string that is long enough to hold a value of type `Str255`. On output, `path` contains the path obtained from the AFP URL specified by the `url` parameter, or `NULL` if `url` was not created with a path.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 54).

DISCUSSION

The `ParseAFPURL` function obtains the values that were used to create an AFP URL.

DisposeAFPURL

Disposes of an AFP URL.

```
void DisposeAFPURL (char * url);
```

`url`	On input, a pointer to a character string that contains an AFP URL previously created by calling `NewAFPURL` (page 49).
function result	None.

DISCUSSION

The `DisposeAFPURL` function releases memory associated with an AFP URL. Your application should call `DisposeAFPURL` when an AFP URL is no longer needed.

Result Codes

The result codes specific to the AFP Client programming interface are listed here.

`kATEnumeratorBadIndexErr`	1	The specified index number is in valid.
`kATEnumeratorBadReferenceErr`	2	The specified enumerator reference is invalid.
`kATEnumeratorBadZoneErr`	3	The specified AppleTalk zone could not be found.
`kATEnumeratorBadPortErr`	4	The port number is not correct.
`kATAppleShareNotAvailableErr`	5	The server is not accepting connections.
`kATServerNotFoundErr`	6	The server could not be located.

AFP and PAP URL Formats

This section describes the format of URLs for Apple services. Items shown in square brackets ([]) are optional. Items shown in *italics* are variable names.

Apple Filing Protocol URLs

This section describes Apple Filing Protocol (AFP) URLs.

For both types of URLs (AppleTalk and TCP), the volume name is optional. If the volume name is not specified, the resolver should query the server for a list of exported volumes and present the list to the user for selection.

For both types of AFP URL, the following user authentication module (UAM) names are valid for the *authtype* portion of the URL:

- `‘No User Authent’` (used for “Guest” login)
- `‘Cleartxt Passwrd’` (the password is passed from the client to the server as clear text.
- `‘Randum Exchange’`
- `‘2-Way Randnum’`
- `‘DHCAST128’`

If the client specifies an *authtype* whose value is `;AUTH=*`, the client may select any authentication type supported by both client and server.

If the client supplies a user name but does not supply an authentication type, the client should use the most secure UAM supported by both client and server. For the current Macintosh resolver, the most secure UAM is “2-Way Randnum Exchange”, which requires a password.

If the client does not specify a user name and authentication type, the “No User Authent” authentication type is used.

If the *authtype* specifies a UAM that the server does not support, the resolver should return `UAM_NOT_SUPPORTED`, but it may also continue the authentication process by using the most secure UAM supported both client and server.

For both types of AFP URL, if the *name*, *password*, or *authtype* portions of the URL contains unsafe or reserved characters, such as double quotation marks (“ “) or semicolons (;), the characters must be encoded as described in RFC 1738.

AppleTalk AFP URL Format

The format of an AppleTalk AFP URL is:

```
afp:/at/[username[;AUTH=authtype][:password]@]server_name[:zone]/
[volumename][/path]
```

TCP/IP AFP URL Format

The format of a TCP/IP AFP URL is:

```
afp://[username[;AUTH=authtype][:password]@]server_name[:port]/
[volumename][/path]
```

Printer Access Protocol URLs

This section describes Printer Access Protocol (PAP) URLs. Currently, no UAM names are defined for the *authtype* portion of PAP URLs.

AppleTalk PAP URL Format

The format of an AppleTalk PAP URL is:

```
pap:/at/[username[;AUTH=authtype][:password]@]printer_name[:zone][/path]
```

TCP/IP PAP URL Format

The format of a TCP/IP PAP URL is:

```
pap://[username[;AUTH=authtype][:password]@]printer_name[:port][/path]
```

Mount Flags

This appendix describes the mount flags that can be provided as the `inMountFlags` and `inAltFlags` parameter to the `AFPMountURL`(page 50) function and `AFPMountSharedVolumeOnMP`(page 33).

The value of the `inMountFlags` or `inAltFlags` parameter can be a combination of the following values:

`MNT_RDONLY` The file system is mounted as read-only so that no one can write to it.

`MNT_NOATIME` The access time on files in the mounted file system are not updated unless the modification time or status change time is also updated.

`MNT_NOEXEC` Executable files in the mounted file system are not allowed to be executed.

`MNT_NOSUID` When files in the mounted file system are executed, the setuid and setgid bits are ignored.

`MNT_NODEV` Special files in the mounted file system are not interpreted.

`MNT_UNION` Files under the mount point are visible instead of hidden.

`MNT_SYNCHRONOUS` All I/IO to the mounted file system should be done synchronously.

`MNT_UPDATE` The mount command is being applied to a file system that is already mounted, thereby allowing mount flags to be changed without unmounting and remounting the file system. Some file systems may not allow all flags to be changed. For example, most file systems do not allow a change from read-write to read-only.

Index

A, B

AFPChangePassword function 37
AFP Client functions
  AFPChangePassword 37
  AFPCreateSharedVolumeEnumerator 25–28
  AFPCreateSVEFromAddress 28–29
  AFPDeleteSharedVolumeEnumerator 37
  AFPGetIndexedSharedVolume 30–31
  AFPGetLoginInformation 34–35
  AFPGetMountAtStartup 35
  AFPGetSharedVolumesCount 29–30
  AFPLibraryPresent 24
  AFPLibraryVersion 24–25
  AFPMountSharedVolume 32
  AFPMountSharedVolumeOnMP 33–34
  AFPSetMountAtStartup 36
  AFPSortSharedVolumes 31
AFP commands
  FPAddIcon 16
  FPGetAuthMethods 14
  FPGetSrvrMsg 18, 19
  FPLogin 14
  FPLoginExt 14
  FPLogout 21
  FPWrite 12, 16
  FPWriteExt 16
AFPCreateSharedVolumeEnumerator function 25–28
AFPCreateSVEFromAddress function 28–29
AFPDeleteSharedVolumeEnumerator function 37
AFPGetIndexedSharedVolume function 30–31
AFPGetLoginInformation function 34–35
AFPGetMountAtStartup function 35
AFPGetSharedVolumesCount function 29–30
AFPLibraryPresent function 24
AFPLibraryVersion function 24–25
AFPMountSharedVolume function 32
AFPMountSharedVolumeonMP function 33–34
AFPMountURL function 50
AFPSetMountAtStartup function 36
AFPSortSharedVolumes function 31
AFP URLs
  format of 56
  functions
    AFPMountURL 50
    IsAFPURL 51
    NewAFPURL 49–50
    ParseAFPURL 52–53
AFPUserBytes 17
application-defined routines
  filter callback routine 40
  notification callback routine 39
  system event callback routine 40–41
ASP commands 13
ASPWriteContinue command 13
attention code bits 18–20
Attention packets 17
attention quantum 15, 16

C

callback routines
  filter 40
  notification 39
  system event 40–41
closing a DSI session 47
codes, result 54
commands
  ASP 13
  ASPWriteContinue 13
  DSAttention 13, 16
  DSCloseSession 13, 21
  DSCommand 13, 15
  DSGetStatus 13, 14

commands *(continued)*
  DSOpenSession 12, 13, 14–15
  DSTickle 13, 16
  DSWrite 12, 13, 16
counting shared volumes 29
creating
  AFP URLs 49
  enumerators 25, 28

D

Data Stream Interface. *See* DSI
deleting
  AFP URLs 53
  enumerators 37
demultiplexing 11
Don't Reconnect bit 18
DSAttention command 13, 16
DSCloseSession command 13, 21, 47
DSCommand command 13, 15, 44–45
DSGetStatus command 13, 14, 42
DSI
  commands
    DSAttention 16
    DSCloseSession 13, 21, 47
    DSCommand 15, 44–45
    DSGetStatus 13, 42
    DSOpenSession 14–15, 43–44
    DSTickle 16
    DSWrite 16, 46–47
  commandsDSAttention 13
  commandsDSCommand 13
  commandsDSTickle 13
  commandsDSWrite 13
  header 11–13
  overview 10–11
DSOpenSession command 12, 13, 14–15, 43–44
DSTickle command 13, 16
DSWrite command 12, 13, 16, 46–47

E

enumerators
  creating 25, 28
  deleting 37

F

filter callback routine 40
FPAddIcon command 16
FPGetAuthMethods command 14
FPGetSrvrMsg command 18, 19
FPLogin command 14
FPLoginExt command 14
FPLogout command 21
FPWrite command 12, 16
FPWriteExt command 16

G, H

getting server messages 18, 19

I, J, K

IPX/SPX 10
IsAFPURL function 51

L

login information, getting 34
long attention messages 20

M

maximum transmission unit 11
mount flags 59

mount information
  getting 35
  setting 36
mounting
  AFP URLs 50
  shared volumes 32, 33
MTU 11

N

NBP 10
`NewAFPURL` function 49–50
notification callback routine 39

O

opening a DSI session 43
Option fields 15

P, Q

packet type 12
PAP URLs 56
`ParseAFPURL` function 52–53
parsing AFP URLs 52
password, changing 37
port number 10

R

reply packets 12
request IDs 12
request packets 12
result codes 54

S

SAP 10
sending DSI commands 44
Server Crash bit 18
Server Message bit 18
server messages, getting 18, 19
server request quantum 15
Service Advertisement Protocol 10
shared volumes
  counting 29
  mounting 32, 33
  names of, getting 30
  sorting by name 31
Shutdown bit 18
sorting shared volume names 31
status information, getting 42
system event callback routine 40–41

T, U

TCP 10
timeouts, detecting 16

V

verifying AFP URLs 51
version, determining 24

W, X, Y, Z

writing data 16