AppleShare IP 6.2
Developer's Kit

User Authentication Modules

Technical Publications

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

About This Manual

This document describes version 2.0 of the application programming interface for client user authentication modules (UAMs). UAMs allow AppleTalk Filing Protocol (AFP) clients to be authenticated with an AppleShare IP server using an alternate authorization scheme, such as Kerberos, Network Information Service (NIS), Windows NT domains, or Novell Directory Services (NDS). For example, an NIS UAM could authenticate a user for a connection to an AppleShare IP file server, mail server, or web server by accessing a central database of user names and passwords stored on an NIS server running on a Sun workstation. Such centralized authentication information would substantially reduced the effort that would otherwise be required to maintain multiple repositories of authentication information.

A UAM implementation consists of a client UAM and a server UAM. This manual describes the method by which a client UAM communicates with a server UAM to authenticate AFP clients. Segments of sample code are included to help developers understand how to use the various calls.

Conventions Used in This Manual

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

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

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

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

For more information

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

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

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

- *AppleShare IP 6.2 Developer's Kit*: AppleShare Registry Library. Apple Computer, Inc.

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

- *AppleShare IP 6.2 Developer's Kit*: AppleTalk Filing Protocol. Apple Computer, Inc.
- *AppleShare IP 6.2 Developer's Kit*: AppleTalk Filing Protocol Version 2.1 and 2.2. Apple Computer, Inc.
- *Inside AppleTalk*, Second Edition. Apple Computer, Inc.

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

- *AppleShare IP 6.2 Developer's Kit*: Server Control Calls and Server Event Handlng. Apple Computer, Inc.

For information on using an AppleShare IP 6.2 file server and Macintosh File Sharing, see the following manuals:

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

CHAPTER 1

User Authentication Modules

This chapter describes the mechanism by which AppleShare IP 6.1 supports third-party user authentication modules (UAMs). Third-party UAMs allow AppleShare IP servers to participate in networks that use an alternative authorization scheme, such as Kerberos, Network Information Service (NIS), Windows NT domains, or Novell Directory Services (NDS).

UAMs can be invoked under the following circumstances:

- When the user uses the Chooser to connect to an AppleShare file server or to connect to a another volume shared by a server to which the client is already connected.
- When an application calls `PBVolumeMount` and specifies the UAM by its protocol name
- When a client mail application connects to an AppleShare mail server.
- When an FTP application connects to an AppleShare FTP server.
- When a web browser connects to an AppleShare web server that is configured to require user authentication.

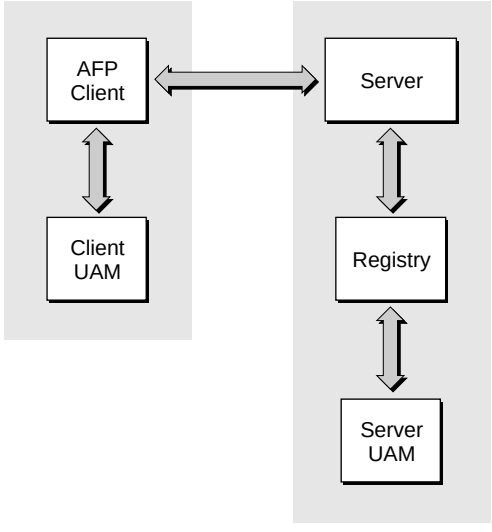
A UAM implementation consists of two parts:

- A server UAM that authenticates users. A server UAM is a PowerPC Code Fragment Manager (CFM) library loaded by the AppleShare Registry at system startup time and called at deferred task time.
- A client UAM that requests a user authentication. A client UAM is a code resource loaded on demand and called at main event time, so the client UAM can use QuickDraw to display dialog boxes and perform other A5-dependent operations.

UAM Architecture

The client UAM and server UAM use the AppleTalk Filing Protocol (AFP) to communicate with each other. Figure 1-1 illustrates the flow of communication between the client UAM and the server UAM.

Figure 1-1 UAM architecture



For an AFP client logging on to an AppleShare file server, the flow of communication between the client and server UAMs occurs in the following sequence:

1. The AFP client calls `AFPServerInfo` in order to determine which UAMs the AFP server supports.
2. If the server supports more than one UAM, the AFP client displays a list of authentication methods for the user to choose from.
3. If the user chooses an authentication method that requires a UAM, the AFP client loads and starts the selected client UAM.
4. Using a callback mechanism to the AFP client, the client UAM opens a session with the AFP server and passes a UAM request that identifies the UAM.
5. The AFP server passes the UAM request to the AppleShare Registry.
6. The AppleShare Registry calls the server UAM and passes the UAM request as a parameter.
7. The server UAM calls the Registry to obtain the user's password and authenticates the user.

8. The server UAM passes the authentication result to the Registry, which returns the result to the AFP server.
9. The AFP server returns the authentication result to the AFP client.
10. The client and server UAM may continue to exchange messages in this way until the server UAM is satisfied

For more information about server UAMs, see Chapter 2, “Server User Authentication Modules.” For more information about client UAMs, see Chapter 3, “Client User Authentication Modules.”

CHAPTER 2

Server User Authentication Modules

This chapter describes the AppleTalk Filing Protocol (AFP) server user authentication module (UAM) interface. A server UAM is a Code Fragment Manager (CFM) library loaded by the AppleShare Registry at system startup in order to perform user authentication.

Server UAMs must meet the following requirements:

- Have a creator code of `'asda'` and type code of `'asru'`.
- Export the following symbols:

```
unsigned long UAMVersion = 0;
unsigned long UAMFlags = 0;
Str16 *UAMName = "\p<unique-uam-name>";     // The protocol name
                                            // of the UAM
```

- Reside in a folder named "AppleShare IP UAM" in the Extensions folder inside the System Folder.
- Implement and export a `UAMAuthenticate` routine that, at minimum authenticates the user's connection request. The `UAMAuthenticate` routine can optionally perform these additional tasks:
  - Change the user's password
  - Change to a specified user ID for a session
  - Create a new user

In addition to implementing a `UAMAuthenticate` routine, server UAMs may

- provide an `_Initialize` Code Fragment Manager routine that initializes the server UAM
- provide a `UAMInitialize` routine that initializes the server UAM
- provide an `_Terminate` Code Fragment Manager routine that prepares the server UAM for shutdown

For additional information about these routines, see "Application-Defined Routines" (page 24).

The server UAM programming interface consists of the following functions:

- `UAMChangeID` (page 2-17), which changes the user ID for a session
- `UAMCreateObject` (page 2-18), which adds a user to the Users & Groups Data File
- `UAMGetAttribute` (page 2-18), which gets the value of a user attribute

- `UAMGetAttributeID` (page 2-20), which gets the value of a user attribute by its object ID
- `UAMSetAttribute` (page 2-21), which sets the value of a user attribute
- `UAMSetAttributeID` (page 2-22), which sets the value of a user attribute by specifying its object ID
- `UAMGetThreadID` (page 2-21), which gets the thread ID of the current thread
- `UAMSleep` (page 2-23), which yields time
- `UAMWakeup` (page 2-23), which wakes up a thread after yielding time

For additional information about these routines, see "Server UAM Functions" later in this chapter.

Note
UAMs are loaded when the computer starts up. Changes to the files in the AppleShare IP UAM folder do not take effect until the next time the computer restarts. ◆

Server UAM Functions

Server UAMs can call the functions described in this section to communicate with the AppleShare Registry.

UAMChangeUID

Change the user ID for a session.

```
extern UInt32 UAMChangeUID (UInt32 newID);
```

`newID` The new user ID.

function result A result code. For a list of possible values, see "Result Codes" (page 27).

DISCUSSION

The `UAMChangeUID` function changes the user ID for the session to the user ID specified by `newID`. The value of `newID` must be a valid user ID in the Users & Groups Data File.

UAMCreateObject

Create an object in the Users & Groups Data File.

```
extern OAMStatus UAMCreateObject(OAMObjectSpec* object)
```

`object` An `OAMObjectSpec` structure that describes the type of object that is being created (such as a user object). For more information, see the *AppleShare Registry Library* in the *AppleShare IP 6.1 Developer's Kit*.

function result A result code. For a list of possible values, see "Result Codes" (page 27).

DISCUSSION

The following sample code creates a user object:

```
OAMObjectSpecaOAMObjectSpec;
    aOAMObjectSpec.objectType = kUser;
    aOAMObjectSpec.specType = kOAMObjectSpecByNameType;
    char *userName = "\pRealUser";
    memcpy(&(aOAMObjectSpec.u.name)userName,strlen(userName));

    aOAMStatus = UAMCreateObject(&aOAMObjectSpec);
```

UAMGetAttribute

Obtain the value of an attribute.

```
extern UInt32 UAMGetAttribute
                    (OAMObjectSpec *spec,
                    OSType creator,
```

```
                    OSType type,
                    void *buffer,
                    UInt32* size);
```

`spec`	Specifies the `OAMObjectSpec` for which the value of an attribute is to be obtained. For information about the `OAMObjectSpec` structures, see *The AppleShare Registry Library,* which is provided as part of the *AppleShare IP 6.1 Developer's Kit*.
`creator`	Specifies the creator code of the attribute whose value is to be obtained. For information about the attribute creator codes defined by Apple Computer, see *The AppleShare Registry Library,* which is provided as part of the *AppleShare IP 6.1 Developer's Kit*.
`type`	Specifies the type code of the attribute whose value is to be obtained. For information about the attribute type codes defined by Apple Computer, see *The AppleShare Registry Library,* which is provided as part of the AppleShare IP Developer's Kit.
`buffer`	On output, contains the value of the attribute identified by `spec`, `creator`, and `type`.
`size`	On input, specifies the length of `buffer`. On output, specifies the length of the data in `buffer`.
function result	A result code. For a list of possible values, see "Result Codes" (page 27).

DISCUSSION

The `UAMGetAttribute` function obtains the value of the attribute identified by the value of the `spec`, `creator`, and `type` parameters.

In the following code sample, the UAM calls `UAMGetAttribute` to obtain the user name attribute:

```
STr32   userName;
UAMGetAttribute(id,kUser,kName,&userName,sizeof(Str32));
```

UAMGetAttributeID

Obtain the value of an attribute by specifying its object ID.

```
extern UInt32 UAMGetAttribute (
                    UInt32 id,
                    OSType creator,
                    OSType type,
                    void *buffer,
                    UInt32* size);
```

`id`	A registry object ID obtained by searching the AppleShare Registry by name and type. For information about attribute names and types, see *The AppleShare Registry Library*, which is provided as part of the *AppleShare IP 6.1 Developer's Kit*.
`creator`	Specifies the creator code of the attribute ID whose value is to be obtained. For information about the attribute creator codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the *AppleShare IP 6.1 Developer's Kit*.
`type`	Specifies the type code of the attribute ID whose value is to be obtained. For information about the attribute type codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the AppleShare IP Developer's Kit.
`buffer`	On output, contains the value of the attribute identified by the `id`, `creator`, and `type` parameters.
`size`	On output, specifies the length of the data in `buffer`.
function result	A result code. For a list of possible values, see "Result Codes" (page 27).

DISCUSSION

The `UAMGetAttributeID` function obtains the value of the attribute identified by the `objectID`, `creator`, and `type` parameters.

UAMGetThreadID

Obtain the current thread's thread ID.

```
extern UInt32 UAMGetThreadID (void);
```

function result The thread ID of the current thread.

DISCUSSION

The `UAMGetThreadID` function return the thread ID of the current thread for subsequent use in calling `UAMWakeup` (page 23).

UAMSetAttribute

Set the value of an attribute.

```
extern UInt32 UAMSetAttribute (
                    OAMObjectSpec *spec,
                    OSType creator,
                    OSType type,
                    void *buffer,
                    UInt32* size);
```

`spec` An `OAMObjectSpec` structure that describes the object for which an attribute is to be set.

`creator` Specifies the creator code of the attribute whose value is to be set. For information about the attribute creator codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the AppleShare IP Developer's Kit.

`type` Specifies the type code of the attribute whose value is to be set. For information about the type creator codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the AppleShare IP Developer's Kit.

`buffer` On input, contains the value of the attribute that is to be set.

`size` On input, specifies the length of the data in `buffer`. If no error occurs, on output, `size` contains the amount of data written in bytes.

function result A result code. For a list of possible values, see “Result Codes” (page 27).

DISCUSSION

The `UAMSetAttribute` function sets the value of the attribute identified by the value of the `id`, `creator`, and `type` parameters.

UAMSetAttributeID

Set the value of an attribute by specifying its object ID.

```
extern UInt32 UAMSetAttributeID (
                    UInt32 id,
                    OSType creator,
                    OSType type,
                    void *buffer,
                    int size);
```

`id` A registry object ID obtained by searching the AppleShare Registry by name and type. For information about attribute names and types, see *The AppleShare Registry Library*, which is provided as part of the *AppleShare IP 6.1 Developer’s Kit*.

`creator` Specifies the creator code of the attribute whose value is to be set. For information about the attribute creator codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the AppleShare IP Developer’s Kit.

`type` Specifies the type code of the attribute whose value is to be set. For information about the type creator codes defined by Apple Computer, see *The AppleShare Registry Library*, which is provided as part of the AppleShare IP Developer’s Kit.

`buffer` On input, contains the value that is to be set.

`size` On input, specifies the length of the data in `buffer`. If no error occurs, on output, `size` contains the amount of data written in bytes.

function result A result code. For a list of possible values, see “Result Codes” (page 27).

DISCUSSION

The `UAMSetAttributeID` function sets the value of the attribute identified by the `id`, `creator`, and `type` parameters.

UAMSleep

Yield time to the AppleShare Registry.

```
extern UInt32 UAMSleep (UInt32 msec);
```

`msec`	Specifies in milliseconds the time to sleep.
function result	`NoErr` if not awakened by a call to `UAMWakeup`. If awakened by a call to `UAMWakeup`, `UAMSleep` returns the value with which `UAMWakeup` was called.

DISCUSSION

The `UAMSleep` function gives the AppleShare Registry time to run. You should call `UAMSleep` before you make a network call. When the network call completes, your completion routine should call `UAMWakeup` to wake up the sleeping thread.

If your completion routine calls `UAMWakeup` before it calls `UAMSleep` (for example, when an asynchronous operation completes before you can call `UAMSleep`), `UAMSleep` returns immediately.

UAMWakeup

Wake up a thread that has yielded time.

```
extern void UInt32 UAMWakeup (
                    ThreadID id,
                    UInt32 value);
```

`id`	A thread ID obtained by calling `UAMGetThreadID` (page 21) that identifies the thread that is to be awakened.
`value`	The value returned by the `UAMSleep` call that put the thread specified by `id` to sleep.

function result A result code. For a list of possible values, see “Result Codes” (page 27).

DISCUSSION

The `UAMWakeup` function wakes up a thread that yielded time to the AppleShare Registry due to a previous call to `UAMSleep` (page 2-23). Your server UAM’s completion routine typically calls `UAMWakeup` to awaken a thread that was put to sleep before the server UAM made a call over the network.

Application-Defined Routines

This section describes the server UAM application-defined routines, which are

- `_Initialize`, an optional routine that initializes the server UAM
- `_Terminate`, an optional routine that prepares a server UAM for system shutdown
- `UAMAuthenticate`, a required routine that authenicates users
- `UAMInitialize`, an optional routine that performs initialization tasks

_Initialize Routine

The `_Initialize` routine is a Code Fragment Manager routine that, if exported, performs initialization tasks. The `_Initialize` routine is called at system task time, so it can call the Memory Manager to allocate memory.

Unlike the `UAMInitialize` routine, the `_UAMIntialize` routine cannot call AppleShare Registry functions.

For information about writing your `_Initialize` routine, see *Inside Macintosh: Power PC System Software.*

Note
Your server UAM can have both an `_Initialize` routine and a `UAMInitialize` routine. ◆

_Terminate Routine

The `_Terminate` routine is a Code Fragment Manager routine that, if exported, performs tasks that must be done before the server shuts down, such as deallocating memory. The `_Terminate` routine is called at system task time, so it can call the Memory Manager.

For information about writing your `_Terminate` routine, see *Inside Macintosh: Power PC System Software.*

UAMAuthenticate Routine

Authenticate a user.

```
SInt32 UAMAuthenticate (
                    SInt32 operation,
                    SInt32 id,
                    void* authState,
                    SInt32 authStateSize,
                    void* authData,
                    SInt32 authDataSize,
                    void* authStateOut,
                    SInt32* authStateSizeOut,
                    void* authDataOut,
                    SInt32* authDataSizeOut);
```

`operation` Specifies the authentication stage, which can be `kUAMAuthLogin` or `kUAMAuthLoginContinue`.

`id` Contains the user ID that is being authenticated.

`authState` Contains authentication-stage dependent information specified by the client UAM.

`authStateSize` Specifies in bytes the length of `authState`.

`authData` Contains input data from the `FPLogin` or `FPLoginContinue` command block. For information on the `FPLogin` and `FPContLogin` command block, see *Inside AppleTalk*, second edition.

`authDataSize` Specifies in bytes the length of `authData`.

`authStateOut` On output, contains at most 16 bytes of authentication-stage–dependent information.

`authStateSizeOut`
Specifies in bytes the length of `authStateOut`.

`authDataOut` Contains a reply message from the server UAM that is to be passed to the client UAM.

`authDataSizeOut`
On input, specifies in bytes the size of `authDataOut`; on output, `authDataSizeOut` specifies the length of the reply message returned in `authDataOut`.

DISCUSSION

Every server UAM must export a `UAMAuthenticate` routine. Before calling a server UAM's `UAMAuthenticate` routine, the AppleShare Registry verifies that the user and the specified UAM exist.

The `UAMAuthenticate` routine is called at deferred task time, so it cannot call the Memory Manager to allocate memory, but it can use other memory allocation mechanisms, such as the Open Transport memory allocation functions, which use the Apple Shared Library Manager.

Note
If your authentication method requires multiple steps, you can use the `authState` parameter to maintain state-dependent information. ◆

UAMInitialize Routine

Initialize a server UAM.

```
Boolean UAMInitialize (void);
```

result If the `UAMInitialize` routine completes successfully, it should return `TRUE`; otherwise, it should return `FALSE`.

DISCUSSION

Server UAMs may export a `UAMInitialize` routine that performs initialization operations, such as allocating memory.

A server UAM's `UAMInitialize` routine is called once during the startup process at deferred task time, so it cannot call the Memory Manager to allocate memory, but it can use other memory allocation mechanisms, such as the Open Transport memory allocation functions, which use the Apple Shared Library Manager.

Server UAM initialization routines are called after the AppleShare Registry is initialized but before the AppleShare Registry is made available for general use. Unlike the `_Initialize` routine, the `UAMIntialize` routine can call AppleShare Registry functions.

If a `UAMInitialize` routine encounters an error, it should return `FALSE`. When a server UAM's `UAMInitialize` routine returns `FALSE`, it is unloaded immediately.

Note
Your server UAM can have both a `UAMInitialize`e routine and an `_Initialize` routine. ◆

Result Codes

Server UAM functions return AppleShare Registry result codes, which are listed here.

`noErr`	0	No error
`kOAMErrInitializationError`	–29300	The AppleShare Registry Library has not been initialized.
`kOAMErrParameterError`	–29301	A parameter is invalid.
`kOAMErrGeneralError`	–29302	An internal error occurred.
`kOAMErrObjectNotFound`	–29310	The specified object or object type does not exist in the Registry.
`kOAMErrContainerObjectNotFound`	–29311	The specified group object does not exist in the Registry.
`kOAMErrMemberObjectNotFound`	–29312	The specified group member does not exist in the Registry.

kOAMErrDuplicateObject	–29320	The specified object already exists in the Registry.
kOAMErrMaximumObjects	–29330	The user object already contains the maximum number of group members.
kOAMErrMaximumMemberObjects	–29331	The group object already has the maximum number of members.
kOAMErrAttributeNotFound	–29340	The specified attribute does not exist in the Registry.
kOAMErrAttributeReadOnly	–29341	The specified attribute allows only read access. Its value is maintained by the Registry.
kOAMErrAttributeReadWriteOnly	–29342	The specified attribute is a required attribute that cannot be deleted.
kOAMErrAttributeBufferTooSmall	–29343	The specified buffer is too small to store the data that has been returned by an AppleShare Registry Library function.
kOAMErrAttributeBufferTooLarge	–29344	The specified buffer is too large to store the data that has been passed to an AppleShare Registry Library function.
kOAMErrMaximumAttributes	–29345	More than 20 attributes have been specified.
kOAMErrBufferTooSmall	–29350	The specified buffer is too small to store the data that has been returned by an AppleShare Registry Library function.
kOAMErrBufferTooLarge	–29351	The specified buffer is too large to store the data that has been passed to an AppleShare Registry Library function.
kOAMErrAuthenticationError	–29360	An authentication error. For example, the specified password is incorrect or the user is not an administrator.

kOAMErrAuthenticationInProgress	–29361	The call to OAMAuthenticate was successful, but additional calls to OAMAuthenticate must be made to complete the authentication process.
kOAMErrLoginDisabled	–29362	Log-on privileges for the user that was used to authenticate this session have been disabled.
kOAMErrAuthenticationServerError	–29363	The server failed a key challenge from the client.
kOAMErrUAMNotFound	–29364	The requested user authentication module does not exist.
kOAMErrAdminDisabled	–29365	Administrative privileges for the user object used to authenticate this session have been disabled.
kOAMErrAuthenticationAdminError	-29366	Administrator authentication failed.
kOAMErrPasswordNeedsChange	–29370	Authentication was successful, but the password of the user object used to authenticate this session must be changed before it can be used again.
kOAMErrPasswordExpired	–29371	Authentication failed. The user's password has expired.
kOAMErrPasswordMinimumLen	–29372	Authentication succeeded, but the password is shorter than the minimum allowed.
kOAMErrSamePassword	–29373	The password specified in a call to OAMChangeObjectKey is the same as the current password.
kOAMErrPasswordChangeDisabled	–29374	The user object specified in a call to OAMChangeObjectKey is not allowed to change the password attribute.
kOAMErrServerNotFound	–29380	The specified agent was not found on the network.

`kOAMErrServerNotInstalled`	–29381	The AppleShare Registry Agent is not installed on this machine.
`kOAMErrServerNotReady`	–29382	The agent is starting up. Reissue the call after a short delay.
`kOAMErrNoMachineName`	–29383	The machine name is not available to the local agent.
`kOAMErrRequestTooLarge`	–29384	The call returned more than the maximum amount of allowable data. Adjust parameters to return less data.
`kOAMErrNetworkError`	–29385	The connection to the Registry has been lost because of a network failure or the termination of an agent. Establish another session when the network is restored and the agent is available.
`kOAMErrSessionIDError`	–29386	The session ID is invalid.
`kOAMErrMaximumSessions`	–29387	Your application tried to open more sessions than it specified when it called `OAMInitialize`.

CHAPTER 3

Client User Authentication Modules

Client user authentication modules (UAMs) are used by AppleTalk Filing Protocol (AFP) clients to implement custom user authentication methods for connecting to and authenticating with an AFP server.

Currently, a UAM is called when the following actions occur:

- The user uses the Chooser to log on to an AFP server that supports the UAM that the user has selected.
- The user is already connected to an AFP server and is using the Chooser to connect to another volume made available by that AFP server.
- A program calls `PBVolumeMount` and specifies that a particular UAM is to be used.

Client UAMs must implement a `UAMCall` routine that can be called by an AFP client or by any other application that needs to authenticate a user. The `UAMCall` routine must implement the following commands:

- `UAMOpen`, to open a session with an AFP server
- `UAMLogin`, to log on to an AFP server
- `UAMClose`, to close a session with an AFP server

Client UAMs can optionally implement the following additional commands:

- `UAMPWDlog`, to display a dialog box that allows the user to enter his or her password
- `UAMVSDlog`, to display a dialog box that allows the user to select the volumes he or she wants to connect to
- `UAMChgPassDlg`, to display a dialog box that allows the user to enter a new password
- `UAMChgPass`, to send a command to the server UAM to change the user's password
- `UAMGetInfoSize`, to get the size of persistent authentication information
- `UAMGetInfo`, to get the persistent authentication information for a connection to a particular AFP server

IMPORTANT

The `UAMCall` routine is always called at system task time. ▲

Client UAMs use callback routines to communicate with an AFP client. The AFP client makes following callback routines available:

- `GetClientInfo`, to obtain information about what the client, such as the versions of AFP the client supports, Gestalt values, and the default user name
- `OpenSession`, to open a session with a server
- `SendMessage`, to send a message to a server once a session has been opened with that server
- `CloseSession`, to close a session with a server
- `SetMic`, to set the message integrity code key
- `EventProc`, to handle events that the client UAM does not handle

UAM files reside in the AppleShare Folder inside the System Folder and have a type code of `'uams'`.

Setting bit 12 (`gestaltAFPClientUAMv2`) of the high word of the `'afps'` Gestalt response indicates that an AFP client supports the UAM interface described in this chapter.

Constants and Data Types

UAMArgs Structure

The `UAMArgs` structure is the only parameter to the `UAMCall` function. The fields of the `UAMArgs` structure define the command type and provide all of the information necessary for `UAMCall` to complete the command successfully.

```
struct UAMArgs {
    short command;
    short sessionRefNum;
    long result;
    void *uamInfo;
    long uamInfoSize;
    ClientUAMCallbackRec *callbacks;
    union {
```

```
        struct UAMChgPassBlk chgPass;
        struct UAMVSDlogBlk vsDlog;
        struct UAMAuthBlk auth;
        struct UAMPWDlogBlk pwDlg;
        struct UAMOpenBlk open;
    };
};
```

Field descriptions

`command` On input, the UAM command code, which must be one of the following values:

```
enum {
  kUAMOpen = 0,
  kUAMPWDlog,
  kUAMLogin,
  kUAMVSDlog,
  kUAMChgPassDlg,
  kUAMChgPass,
  kUAMGetInfoSize,
  kUAMGetInfo,
  kUAMClose,
  kUAMPrOpen,
  kUAMPrAuthDlog,
  kUAMPrAuth
};
```

`sessionRefNum` An AFP session reference number. If an AFP session is not already in progress, an AFP session reference number is returned by the client UAM during the `UAMLogin` call. If an AFP session is in progress, the AFP session reference number is passed during the `UAMOpen` call and all subsequent calls for a particular session.

`result` On output, an `OSStatus` reflecting the result of calling `UAMCall` with a particular UAM command code. Typical values are `noErr`, `userCancelledError`, `afpUserNotAuthErr`, `afpPwdTooShortErr`, `afpPwdExpiredErr`, and `afpPwdNeedsChangeErr`.

`uamInfo` On input, a pointer to the buffer (allocated by the AFP client in the system heap) in which the `GetUAMInfo` call (page 3-46) is to store persistent authentication information. When logging in via the Chooser, the `uamInfo` field is nil

	until the AFP client calls `UAMCall` with a command of `GetUAMInfo`. All other UAM commands should treat this field as a read-only field. The AFP client is responsible for disposing of the buffer pointed to by `uamInfo`.
`uamInfoSize`	On input, the size in bytes of `uamInfo`. On output, `UAMCall` sets `uamInfoSize` to reflect the current size of `uamInfo`.
`callbacks`	On input, a pointer to the `ClientUAMCallbackRec` structure (page 3-35) for this session.
union	If the value of `command` is `kUAMChgPass` or `kUAMChgPassDlg`, on input,*union* is a `UAMChgPassBlck` structure (page 3-36).
	If the value of `command` is `kUAMVSDlog`, on input,*union* is a `UAMVSDlogBlk` structure (page 3-36).
	If the value of `command` is `kUAMLogin`, on input.*union* is a `UAMAuthBlk` structure (page 3-37).
	If the value of `command` is `kUAMPWDlog`, on input,*union* is a `UAMPWDlogBlk` structure (page 3-37).
	If the value of `command` is `kUAMOpen`, on input,*union* is a `UAMOpenBlk` structure (page 3-38).

ClientUAMCallbackRec Structure

The `ClientUAMCallbackRec` structure is a field in the `UAMArgs` structure used to store pointers to callback routines. UAMs written for PowerPC-based Macintosh computers must use the `CallUniversalProc` routine to call the UAM callback routines; UAMs written for 68K -based Macintosh computers jump to the callback routines as if they were function pointers.

```
struct ClientUAMCallbackRec {
    UniversalProcPtr    OpenSessionUPP;
    UniversalProcPtr    SendRequestUPP;
    UniversalProcPtr    CloseSessionUPP;
    UniversalProcPtr    GetClientInfoUPP;
    UniversalProcPtr    SetMicUPP;
    UniversalProcPtr    EventProcUPP;
};
```

Field descriptions

`OpenSessionUPP`	A pointer to an AFP client's `OpenSession` callback routine (page 3-48).
`SendRequestUPP`	A pointer to an AFP client's `SendRequest` callback routine (page 3-49).
`CloseSessionUPP`	A pointer to an AFP client's `CloseSession` callback routine (page 3-49).
`GetClientInfoUPP`	A pointer to an AFP client's `GetClientInfo` call back routine (page 3-47).
`SetMicUPP`	A pointer to an AFP client's `SetMic` callback routine (page 3-50).
`EventProcUpp`	A pointer to an AFP client's `EventProc` callback routine (page 3-50).

UAMChgPassBlk Structure

The `UAMChgPassBlk` structure is passed as a field in a `UAMArgs` structure when the value of `UAMArgs.command` is `kUAMChgPass` or `kUAMChgPassDlg`.

```
struct UAMChgPassBlk {
    StringPtr userName;
    StringPtr oldPass;
    StringPtr newPass;
};
```

Field descriptions

`userName`	On input, a pointer to a string that contains the user name.
`oldPass`	On input, a pointer to a string that contains the password being changed.
`newPass`	On input, a pointer to a string that contains the new password.

UAMVSDlogBlk Structure

The `UAMVSDlogBlk` structure is passed as a field in a `UAMArgs` structure when the value of `UAMArgs.command` is `kUAMVSDlog`.

```
struct UAMVSDlogBlk {
    short numVolumes;
    VolListElem *volumes;
};
```

Field descriptions

`numVolumes`	On input, the number of volumes in `volumes`.
`volumes`	On input, a `VolListElem` structure (page 3-40) that lists the volumes the server makes available for mounting.

UAMAuthBlk Structure

The `UAMAuthBlk` structure is passed as a field in a `UAMArgs` structure when the value of `UAMArgs.command` is `kUAMLogin`.

```
struct UAMAuthBlk {
    StringPtr   userName;
    UInt8 *     password;
    OTAddress   *srvrAddress;
};
```

Field descriptions

`userName`	On input, a pointer to a 64-byte Pascal string that contains the name of the user who is to be authenticated.
`password`	On input, a pointer to a 64-byte value that contains the user's password.
`OTAddress`	On input, a pointer to an `OTAddress` that contains the address of the server.

UAMPWDlogBlk Structure

The `UAMPWDlogBlk` structure is passed as a field in a `UAMArgs` structure when the value of `UAMArgs.command` is `kUAMPWDlog`.

```
struct UAMPWDlogBlk{
    StringPtr   userName;
    UInt8 *     password;
};
```

Field descriptions

`userName`	A pointer to a 64-byte Pascal string that contains the name of the user who is to be authenticated.
`password`	A pointer to a 64-byte vale that contains the password.

UAMOpenBlk Structure

The `UAMOpenBlk` structure is passed as a parameter to `UAMCall` when `UAMCall` is called with a command code of `UAMOpen`.

```
struct UAMOpenBlk {
    StringPtr       objectName;
    StringPtr       zoneName;
    OTAddress       *srvrAddress;
    SrvrInfoBuffer  *srvrInfo;
};
```

Field descriptions

`objectName`	On input, the name of the server that is to be opened.
`zoneName`	On input, the name of the zone in which the server, or `nil` if there is no zone.
`srvrAddress`	On input, the Open Transport address of the server.
`srvrInfo`	On input, information returned by calling `GetStatus`.

ClientInfo Structure

The `ClientInfo` structure is used to return information about the AFP client to the UAM.

```
struct ClientInfo {
    short       fInfoType;
    StringPtr   fDefaultUserName;
};
```

Field descriptions

`fInfoType` On input, the type of client information. The value of `fInfoType` must be one of the following values:

```
enum {
  kAFPClientInfo = 0,  // Information about the client
of
                       // an AFP server
  kPrClientInfo = 1    // Reserved.
};
```

`fDefaultUserName` On input, a pointer to a string that contains the default user name.

AFPClientInfo Structure

The `AFPClientInfo` structure is used to return information about the version of AFP that an AFP client supports.

```
struct AFPClientInfo {
    short       fInfoType;
    StringPtr   fDefaultUserName;
    short       fConfigFlags;
    short       fVersion;
    short       fNumAFPVersions;
    char        **fAFPVersionStrs;
};
```

Field descriptions

`fInfoType` On input, the type of client information. For an `AFPClientInfo` structure, the value of `fInfoType` must be `kAFPClientInfo`.

`fDefaultUserName` On input, a pointer to a string that contains the default user name.

`fConfigFlags` On input, the high 16 bits of the `'afps'` Gestalt response.

`fVersion` On input, the low 16 bits of the `'afps'` Gestalt response.

`fNumAFPVersions` On input, the number of AFP versions that this client supports.

`fAFPVersionStrs` On input, a handle to an array of strings, each of which describes a version of AFP that this client supports.

VolListElem Structure

The `VolListElem` structure is used in the `UAMVSDlogBlk` structure (page 3-40) to store status information about volumes.

```
struct  VolListElem {
    byte    flags;
    Str32   volName;
};
```

Field descriptions

`flags` A bit field (obtained by calling `GetSrvrParms`) whose values are interpreted by the following enumeration:

```
enum {
    kMountFlag      = 0,    // On output, the UAM sets this bit to
                            // indicate that this volume is to be mounted
    kAlreadyMounted = 1,    // On input, a bit telling the UAM that this
                            // volume is currently mounted
    kHasVolPw       = 7     // On input, a bit telling the UAM that the
                            // volume has a volume password
};
```

`volName` The name of a volume.

UAMMessage Structure

The `UAMMessage` structure is used by the client UAM to pass information back to the AFP client when the client UAM calls the AFP client's `OpenRequest` and `SendRequest` callback routines. A `UAMMessage` structure is also passed as a parameter to the client UAM's completion routine.

```
struct UAMMessage {
    short           commandCode;
    short           sessionRefNum;
    unsigned char   *cmdBuffer;
    unsigned long   cmdBufferSize;
    unsigned char   *replyBuffer;
    unsigned long   replyBufferSize;
    CompletionPtr   *completion;
```

```
    void *contextPtr;
};
typedef struct UAMMessage UAMMessage, *UAMMessagePtr;
```

Field descriptions

`commandCode` A command code. The value of `commandCode` must be one of the following:

```
enum {
  kOpenSession = 'UAOS'
  kSendRequest = 'UASR'
};
```

`sessionRefNum` The session reference number for this session, returned when the value of `commandCode` is `kOpenSession` and passed back in subsequent messages sent via the `OpenSession` callback.

`cmdBuffer` A pointer to a buffer containing an AFP command, such as `afpLogin` or `afpContLogin`, and the command parameters for that command. For a complete list of AFP commands, see *Inside Macintosh: Networking*.

`cmdBufferSize` The length of the command in `cmdBuffer`.

`replyBuffer` A pointer to a buffer that is used to return a reply.

`replyBufferSize` The length of the reply in `replyBuffer`.

`completion` A pointer to a completion routine.

`contextPtr` A pointer to a value that identifies this session. If `contextPtr` is not nil, it is passed to a completion routine when completion routine is called.

Client UAM Routines

UAMCall Routine

Send a command to a server UAM.

```
pascal OSErr UAMCall(UAMArgs *);
```

UAMArgs	A UAMArgs structure whose fields define the command type and provide the information required to complete the call successfully.

If a fatal error occurs for which a client UAM puts up a dialog box, the client UAM should return userCancelledErr to back out of the UAM call.

DISCUSSION

If you are implementing a client UAM, you must implement a UAMCall routine. The AFP client must call UAMCall from it's main event loop so the client UAM can make A5-dependent calls, such as calls to QuickDraw and the Resource Manager.

shows the typical sequence of commands for three scenarios:

Table 3-1 Typcial client UAM command sequence

Chooser login	Chooser already connected	Alias resolution[1]
1. UAMOpen	1. UAMOpen	1. UAMOpen
1a. UAMPWDlog[2]	1a. UAMChgPassDlg[2]	1a. UAMPWDlog[2]
2. UAMLogin	1b. UAMChgPass[2]	2. UAMLogin
2a. UAMChgPassDlg[2]	1c. UAMVSDlog[2]	2a. UAMChgPassDlg[2]
2b. UAMChgPass[2]	1d. UAMGetInfoSize[2]	2b. UAMChgPass[2]
2c. UAMVSDlog[2]	1e. UAMGetInfo[2]	2c. UAMGetInfoSize[2]
2d. UAMGetInfoSize[2]	2. UAMClose	2d. UAMGetInfo[2]
2e. UAMGetInfo[2]		3. UAMClose
3. UAMClose		

[1]This sequence is typical of any program that calls PBVolumeMount specifying the protocol name of the UAM as a parameter.
[2]Optional commands.

As noted in , some client UAM commands are optional. The value returned to the AFP client by your UAM's UAMOpen entry point indicates the optional commands that your UAM supports and determines whether the AFP client will call any optional commands supported by your UAM. The mechanism for

indicating support for optional commands is described in the section “UAMOpen Command” (page 43).

UAMOpen Command

Your UAM’s `UAMCall` routine is called with a command of `UAMOpen` after the AppleShare client loads the clietn UAM’s code resource. The object name, object zone (if available), Open Transport address, and the server information are passed in. If the connection is already established the `sessionRefNum` field is filled in; otherwise the value of the `sessionRefNum` field is 0.

Your UAM must return a 32-bit value named `configInfo`, which the AFP client interprets as an `OSStatus` if its value is less than zero. Otherwise, set the bits in `configInfo` as described in Table 3-2 to indicate the UAM commands that your UAM supports.

Table 3-2 Bit values of `configInfo`

Bit	Meaning
0	Your UAM provides its own password dialog box
1	Your UAM provides its own volume selection dialog box.
2	Your UAM supports change password
3	Your UAM provides its own change password dialog box
4	Your UAM returns information in the `UAMInfo` field of the `UAMArgs` structure. Please see the note that follows.
5 to 31	Reserved and must be set to zero.

Note
If your UAM does not return information in the `UAMInfo` field of the `UAMArgs` structure, the `UAMInfo` pointer is `nil` and the AFP client cannot call your `UAMCall` routine with a command of `UAMGetInfo` or `UAMGetInfoSize`, ◆

UAMPWDlog Command

When your UAM's `UAMCall` routine is called with a command of `UAMPWDlog`, you should display the standard password dialog box for obtaining the user's name and password. A `UAMPWDlogBlk` structure is used to store the user's name and password.

If you already have enough information to authenticate the user, you don't need to display the dialog box.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMPWDlog` only if bit 0 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

UAMLogin Command

Your UAM's `UAMCall` routine is called with a command of `UAMLogin` to connect to the server. The values of the `userName` and `password` fields of the `UAMAuthBlk` structure are the same as the `userName` and `password` fields of the `UAMPWDlogBlk` structure.

Note
Before your UAM's `UAMLogin` routine returns, it must store the session reference number for the session in the `sessionRefNum` field of the `UAMArgs` structure. ◆

UAMChgPassDlg Command

Your UAM's `UAMCall` routine is called with a command of `UAMChgPassDlg` when the user clicks the Change Password button in the standard password dialog box or in the "Already connected" dialog box.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMChgPassDlg` only if bit 3 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

If you implement `UAMChgPassDlg`, you should also implement `UAMChgPass`.

UAMChgPass Command

Your UAM's `UAMCall` routine is called with a command of `UAMChgPass` after calling `UAMCall` with a command of `UAMChgPassDlg` to change the password.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMChgPass` only if bit 2 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

UAMVSDlog Command

Your UAM's `UAMCall` routine is called with a command of `UAMVSDlog` to display the volume selection list. The list does not contain volumes that are already mounted from this server. The bits in the volume flags byte are set from the `GetSrvrParms` reply. To specify that a volume should be mounted, the `kMountFlag` bit in the volume flags must be set.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMVSDlog` only if bit 1 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

Under certain circumstances, the `UAMVSDlog` is not used, such as when Navigation Services builds a volume list. Do not depend on `UAMVSDlog` being used for every volume mount.

UAMGetInfoSize Command

After a successful call to `UAMCall` with a command of `UAMLogin`, your UAM's `UAMCall` routine is called with a command of `UAMGetInfoSize` to obtain the size of the persisitent authentication information for this session.

Your implementation of the `UAMGetInfoSize` command should store the size in bytes of the persisent authentication information in the `uamInfoSize` field of the `UAMArgs` structure.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMGetInfoSize` only if bit 4 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

UAMGetInfo Command

Your UAM's `UAMCall` routine is called with a command of `UAMGetInfo` to get persistent authentication information.

Note
Your UAM's `UAMCall` routine is called with a command of `UAMGetInfo` only if bit 4 is set in the `configInfo` value returned by previously calling `UAMCall` with a command of `UAMOpen`. ◆

Before the AFP client calls `UAMCall` with a command of `UAMGetInfo`, it calls `UAMCall` with a command of `UAMGetInfoSize` to get the size of the persistent authentication information. Then the AFP client allocates a buffer of the appropriate size in the system heap and sets `UAMArgs.uamInfo` to point to it.

Your implementation of the `UAMGetInfo` command should copy the persistent authentication information into the buffer pointed to by `UAMArgs.uamInfo`. The UAM info is part of the `VolMountInfoBlk` returned by the `GetVolMountInfo` call and passed as a parameter to the `PBMountVol` call.

When the client UAM is called by code that implements the `PBVolumeMount` call, `UAMArgs.uamInfo` points to the `UAMInfo` field in the `VolMountInfoBlock` (if that field is present).

In the case of the `PBVolumeMount` call or when the AFP client already has a connection to the server, `UAMArgs.uamInfo` points to a buffer that is of the size returned by `GetVolInfoSize`.

Note
Your implementation of the `UAMGetInfo` command should only copy persistent authentication information—it should not copy volume information. ◆

The persistent authentication information returned by the client UAM is read-only and should not be changed. Its persists until the AFP client calls the client UAM's `UAMClose` command.

The AFP client is responsible for disposing of the buffer that it allocated for storing persistent authentication information.

UAMClose Command

Your UAM's `UAMCall` routine is called with a command of `UAMClose` to close the UAM. Your UAM should deallocate any memory that it has allocated and unload any shared libraries that it may have loaded.

Callback Routines

Client UAMs use callback routines to communicate with an AFP client. The AppleShare Client 3.7 makes available the callback routines described in this section.

EventProc Callback

Passes an event record to an AFP client.

```
(void EventCallbackPtr) (EventRecord *theEvent);
```

DISCUSSION

The `EventProc` callback routine passes an event record to the AFP client. The client UAM should call the `EventProc` callback whenever it receives an event record for an event that does not belong to the client UAM.

GetClientInfo Callback

Returns information about an AFP client.

```
pascal ClientInfo *GetClientInfo(short infoType);
```

`infoType` A value the defines the type of information that is being requested. The value of `infoType` must be one of the following:

```
enum {
  kAFPClientInfo = 0, // Information about the client of
                      // an AFP server
  kPrClientInfo = 1   // Reserved
};
```

DISCUSSION

The `GetClientInfo` callback routine returns information about an AFP client, such as the versions of AFP that it supports, Gestalt values, and the default user name. If the AFP client does not support the `UAMInfo` type, `GetClientInfo` returns `nil`.

OpenSession Callback

Opens a session at the specified address.

```
pascal OSStatus OpenSession(OTAddress *,
                    const char* endpointString,
                    UAMMessagePtr message);
```

`OTAddress` Address of the server.

`endpointString`
The endpoint string for the connection. To specify the default endpoint string, set `endpointString` to `nil`. The endpoint string provides a way to specify streams configuration information on a per-connection basis. It is only used for TCP/IP connections and is ignored for AppleTalk connections.

`message` Pointer to a `UAMMessage` structure (page 3-40).

DISCUSSION

The `OpenSession` callback routine opens a session at the address specified by `OTAddress`. The value of the `commandCode` field in the `UAMMessage` structure must be `kOpenRequest`. The session reference number for the opened session is returned in the `sessionRefNum` the `UAMMessage` structure.

For sessions over AppleTalk, the size of `cmdBuffer` is limited to `kMaxAFPCommand` (576 bytes), `cmdBuffer` must be `afplogin` , and the `endpointString` parameter is ignored.

For synchronous operation, set the `completion` and `contextPtr` fields of the `UAMMessage` structure to `nil`. For asynchronous operation, set the `completion` field of the `UAMMessage` structure to point to your completion routine and set the `contextPtr` field to a value that identifies this request.

SendRequest Callback

Sends a message to a server.

```
pascal OSStatus SendRequest(UAMMessagePtr message);
```

`message` Pointer to a `UAMMessage` structure (page 3-40).

DISCUSSION

The `SendRequest` callback routine sends a command to the server. The value of `UAMMessage.commandCode` must be `kSendRequest`.

For AFP connections, the size of `cmdBuffer` is limited to `kMaxAFPCommand` (576 bytes) and `cmdBuffer` must contain an AFP command.

For synchronous operation, set `UAMMessage.completion` and `UAMMessage.contextPtr` to nil. For asynchronous operation, set `UAMMessage.completion` to point to your completion routine and set `UAMMessage.contextPtr` to a value that identifies this request.

The value of `UAMMessage.sessionRefNum` is the session reference number returned by previously calling the AFP client's `OpenSession` callback routine.

CloseSession Callback

Closes a session with an AFP server.

```
pascal OSStatus CloseSession(short sessRefNum);
```

`sessRefNum` Identifies the session that is to be closed.

DISCUSSION

The `CloseSession` callback routine closes a session with an AFP server.

SetMic Callback

Sets the message integrity code key.

```
pascal OSStatus SetMic(short sizeInBytes,
                       Ptr micValue);
```

`sizeInBytes` The size of `micValue`.

`micValue` The message integrity code key.

DISCUSSION

If the connection supports using keyed HMAC-SHA1 for message integrity, the client UAM can pass a key to the network layer using this call.

Note
This callback is still in development.

Completion Routine

This completion routine is called at interrupt time with the `contextPtr` passed in to the `OpenSession` and `SendRequest` calls, when one of these calls completes. The `result` parameter contains the AFP result. You cannot call any of the callback routines from this completion routine, so you can't do chained completion routines.

```
typedef pascal void (*CompletionPtr)(
                    UAMMessagePtr message,
                    void* contextPtr,
                    OSStatus result);
```

`CompletionPtr` A pointer to the completion routine.

`message` A pointer to a `UAMMessage` structure.

`contextPtr`	A value returned by the previous execution of the AFP client's `OpenSession` or `SendRequest` callback routine.
`result`	An AFP result code indicating the status of the completion routine. See the *AppleTalk Filing Protocol* document in the *AppleShare IP 6.1 Developer's Kit* for the list of result codes.

Resources

For system software versions 7 and 8, a client UAM is a safe fat code resource that allows for 68k and PowerPC UAM implementations.

The `'uamg'` Resource

All UAM files have a `'uamg'` resource whose ID is 0. The `'uamg'` resource is the UAM Info resource and it contains the following information:

```
type 'uamg'
{
    integer     VersionNumber;
    integer     UAMClass;
    integer     PasswordLength;
    byte        PassDlogFlag;
    byte        VolDlogFlag;
    byte        UAMType;
    byte        UReserved;
};
```

Field descriptions

`VersionNumber`	Denotes the version of the UAM API that this UAM conforms to. For version 2.0 of the AFP client UAM interface, `VersionNumber` must be 2.
`UAMClass`	Denotes the class of the UAM. The value of `UAMClass` must be one of the following values: 0 indicates that this UAM uses Apple Computer's current UAM support, which consists of no user authentication, cleartext password, random number exchange, and

	two-way random number exchange. They cannot be replaced. 1 indicates that this class supports cleartext passwords longer than 8 characters. If you use this class, you don't need a `'uamc'` resource because support for this class is built into the client—you only need to implement a server-side UAM. 2 indicates that this class supports encrypted passwords longer than 8 characters. If you use this class, you don't need a `'uamc'` resource because support for this class is built into the client—you only need to implement a server-side UAM. 3 indicates that this UAM uses a UAM-defined authentication method. Use this class if you want to provide your own user interface and write code that handles the login sequence. Code that implements class 3 UAMs is stored as packed `'uamc'` ID 0 resource.
`PasswordLength`	Specifies the maximum password length that the UAM supports. The value of `PasswordLength` can be from 0 to 64.
`PassDlogFlag`	Obsolete. Replaced by the `configInfo` flags returned by `UAMOpen` (page 3-43).
`VolDlogFlag`	Obsolete. Replaced by the `configInfo` flags returned by `UAMOpen` (page 3-43).
`UAMType`	A user-defined ID in the range of 128 to 255. It is returned by the `GetVolParams` call as well as other calls. The AFP client does not depend on the value of `UAMType` to identify a particular UAM; instead, the AFP client uses a UAM's protocol name, as described in "The `'uamn'` Resource" (page 53), to distinguish one UAM from another.
`UReserved`	Reserved. The value of `UReserved` is always zero.

The `'uamc'` Resource

Class 3 UAMs store the code that implements their user interface and logon handling sequence in a packed `'uamc'` resource whose ID is 0.

The 'uamn' Resource

The 'uamn' resource is used to store strings.

```
type 'uamn' as 'STR ';          // UAM string resources
resource 'uamn' (0, "UAM name") // Name shown in UAM select dialog
{
    "Type 2 Class 3 UAM"
};

resource 'uamn' (1, "AFP UAM name") // Protocol name of UAM
{
    "Cleartxt Passwrd"
};

resource 'uamn' (2, "UAM Description string")   // Description shown in
                                                // password dialog
{
    "(Sample UAM)"
};
```

Sample UAM Client

The sample code shown in Listing 3-1 opens a session with an AFP server and logs the user on.

Listing 3-1 Sample client UAM

```
#include <Types.h>
#include "ClientUAM.h"
#include <String.h>
#include <Resources.h>
#include <A4Stuff.h>
#include "SampleUAM.h"
#include "AFPPackets.h"
```

```
enum {
    kSampleCfg = (1 << kUseVolDlog),// The value returned by UAMOpen
};

Boolean FindStringInBuf(StringPtr,Ptr,UInt32);
long    SampleOpen(UAMArgs *theArgs);
OSStatusSampleLogin(UAMArgs *theArgs);

unsigned char commandBuffer[200];
unsigned char replyBuffer[512];
StringPtr gAFPVersion;

StringPtr FigureAFPVersion(AFPSrvrInfo *,ClientUAMCallbackRec *theCallbacks);

pascal OSErr main(UAMArgs *theArgs)
{
    EnterCodeResource();
    OSErr error;
    switch(theArgs->command)
    {
        case UAMOpen:
            error = SampleOpen(theArgs);
            break;

        case kUAMPWDlog:
            error = kNotForUs;
            break;

        case kUAMLogin:
            error = SampleLogin(theArgs);
            break;

        case kUAMVSDlog:
            DebugStr("\pPut up a Volume Select dialog");
            error = noErr;
            break;

        case kUAMChgPassDlg:
            error = kNotForUs;
            break;
```

```
        case kUAMChgPass:
            error = kNotForUs;
            break;

        case kUAMGetInfoSize:
            error = kNotForUs;
            break;

        case kUAMGetInfo:
            error = kNotForUs;
            break;

        case kUAMClose:
            error = NoErr;
            break;

        default:
            error = kNotForUs;
            break;
    }

    ExitCodeResource();
    return error;
}

longSampleOpen(UAMArgs *theArgs)
{
    gAFPVersion = FigureAFPVersion(theArgs->Opt.open.srvrInfo,theArgs->callbacks);
    theArgs->result = kSampleCfg;
    return noErr;
}

OSStatus SampleLogin(UAMArgs *theArgs){
    OSStatus theError = kUAMError
    Ptr cmd;
    unsigned long cmdSize;
    Handle theUAMName;
    UAMMessag message;
    StringPtr user = theArgs->Opt.auth.userName;
    StringPtr password = theArgs->Opt.auth.password;
```

```
    if(!gAFPVersion){
        // Put up an alert and return userCanceled error
        DebugStr("\pno AFP version");
        return userCanceledErr;
    }

    if(theArgs->callbacks)
{
        commandBuffer[0] = kFPLogin;
        cmd = (Ptr) &commandBuffer[1];
        memcpy(cmd,(const char *)&gAFPVersion[0],gAFPVersion[0]+1);
        cmd += gAFPVersion[0] + 1;

        // Get the UAMString from the resource
        theUAMName = Get1Resource(kUAMStr,kUAMProtoName);
        if(!theUAMName)
            return ResError();// Depends on ResLoad being TRUE

        // Put the UAMString into the command buffer
        HLock(theUAMName);
        memcpy(cmd,(const char *)&((*theUAMName)[0]),(*theUAMName)[0]+1);
        cmd += (*theUAMName)[0]+1;
        HUnlock(theUAMName);
        ReleaseResource(theUAMName);

        // Copy in the username
        memcpy(cmd,(const char *)&user[0],user[0]+1);
        cmd += user[0]+1;

        // Test for an odd boundary
        if(((UInt32)cmd - (UInt32)commandBuffer) & 0x01)
        {
            *cmd++ = 0x00;// If an odd boundary, put in some padding
        }

        // Copy in the password (a maximum of 8 bytes)
        memcpy(cmd,(const char *)&password[0],8);
        cmd += 8;

        // Get the size of the command buffer
        cmdSize = (unsigned long)((unsigned long)cmd - (unsigned long)commandBuffer);
```

```
        message.commandCode = kOpenSession;
        message.cmdBuffer = commandBuffer;
        message.cmdBufferSize = cmdSize;
        message.replyBuffer = nil;
        message.replyBufferSize = 0;
        message.completion = nil;
        message.contextPtr = nil;

        //Make the login call.);

        theError =
        theArgs->callbacks->OpenSessionUPP(theArgs->Opt.auth.srvrAddress,nil,&message);
        if(!theError){
            theArgs->sessionRefNum = message.sessionRefNum;
        }
        theError = message.result;

    }
    return theError;
}

StringPtr FigureAFPVersion(AFPSrvrInfo *info,ClientUAMCallbackRec *callbacks);
{
    struct AFPClientInfo *theClientInfo = nil;
    short index;
    Ptr versBuf;
    UInt32 versBufsize;
    GetClientInfoPtr *fcn;

    callbacks->GetClientInfoUPP(kAFPClientInfo,(ClientInfo **)&theClientInfo);

    if(theClientInfo){
    // Go through the list of supported AFP versions and try to find them
    // in the SrvrInfoBuffer. The first match is accepted,
        versBuf = (Ptr)((UInt32)info + info->fVerCountOffset+1);
        versBufsize = kMaxAFPCommand - info->fVerCountOffset;// The largest size

        for(index = 0; index < theClientInfo->fNumAFPVersions; index++){
            if(FindStringInBuf
                    (theClientInfo->fAFPVersionStrs[index],versBuf,versBufsize)){
```

```
                return theClientInfo->fAFPVersionStrs[index];
            }
        }
    }
    return nil;
}

Boolean FindStringInBuf(StringPtr string, Ptr buf, UInt32 bufSize)
{
    Ptr end = buf + bufSize;
    Byte len = string[0] + 1;
    short index;

    while((buf < end) && (*buf++ != string[0])) ; // Scan for the proper length.

    if(!(buf < end)){
        return false;
    }
    for(index = 1; (index < len) && (buf > end); index++){
        if(*buf++ != string[index])
            return false;
    }

    if(!(buf < end)){
        return false;
    }
    return true;
}
```

Index

A, B

C, D

E

F

G, H

I, J, K, L, M, N

O

P, Q

R

S

T

U

V

W, X

Y, Z