

AppleShare File Server 3.0 Control

CHAPTER 1-SERVER CONTROL CALLS

This chapter introduces the server control calls available with the AppleShare File Server 3.0 and describes how server control calls interact with the main elements of file server software. The chapter presents each server control call individually and includes a sample code segment for each call that demonstrates how you might use the call in your own programs.

Server control calls enable applications to monitor and control the major functions of the AppleShare File Server 3.0. These control calls let your programs

- get and modify server configuration information
- check a server's status
- start and stop file service
- get information on users, volumes, and shared items
- disconnect users (including the users of a specific volume)
- send messages to users
- set or clear the copy-protect status of files
- use server event handlers

Server control calls, together with server event handling (described in Chapter 2), make it possible to create any number of services and utilities for the AppleShare File Server 3.0. Because you can monitor file usage -- who uses files, which files are saved to or deleted from a server, where files are copied to, and so on -- you can create file-usage audit trails, generate server-usage statistics, and perform other types of accounting services. You can also control file servers remotely. By monitoring the number of active users, logging off idle users, and controlling log-on access, you can perform load-balancing services for a group of related servers. Many other services are possible. AppleShare File Server 3.0 server control calls and event handling form a complete interface through which your applications and programs can control and extend the capabilities of the file server software. This guide refers to such programs and applications as server additions.

Note Macintosh File Sharing supports a subset of the AppleShare File Server 3.0 server control calls. See Appendix A for a list of these calls.

Main elements of file servers and server control calls

This section describes the software components and data files that make up the AppleShare File Server 3.0 and Macintosh File Sharing. Because the AppleShare File Server 3.0 and Macintosh File Sharing perform similar functions, the components for each are similar and both use the same types of data files.

AppleShare File Server 3.0 software components

The AppleShare File Server 3.0 is composed of a number of files. The File Server Extension provides the actual functionality of the file server. The AppleShare File Server and the AppleShare Admin applications provide the user interface for the server.

This section describes each of AppleShare File Server 3.0 software components. The section "Data Files," later in this chapter, describes the Users & Groups Data File and the AppleShare PDS file.

File Server Extension The File Server Extension contains the actual file server code. It is an extension of the system and resides in the Extensions folder. The File Server Extension is a launchable file, though its file type is 'INIT' instead of 'APPL', which prevents users from starting it from the Finder. (The 'INIT' file type also tells the system to put the file in the Extensions folder and causes the extension to be opened during system startup.) When the File Server Extension is launched, it runs as a background application.

The File Server Extension contains no user interface of its own. The user interface is provided by the AppleShare File Server and AppleShare Admin applications (described next). These applications communicate with the File Server Extension primarily by means of server control calls. Server control calls are also the primary means of communication between server additions and the file server. The File Server Extension communicates with remote AppleShare clients through Apple Filing Protocol (AFP) sessions, and, locally, with shared volumes and files by means of Macintosh File Manager routines.

When the File Server Extension is launched, it checks its environment, the Users & Groups Data File, and the desktop databases and AppleShare PDS files of appropriate volumes. (The File Server Extension does not attempt to share remote volumes, or volumes such as floppy disks or volumes that are ejected and off line during startup.) If an important required condition is not satisfied, the offending volume will not be prepared for use with the file server or the file server will not be enabled.

Once started, the File Server Extension takes over the dispatching of all file system calls -- both local calls and remote requests. Essentially, the file server acts as a mediator between the network and your local HFS volumes. The file server imposes access privilege constraints on AFP requests and implements some calls that are not implemented in HFS -- such as those that govern byte-range locking, access privileges, and extended file access permissions.

AppleShare File Server The AppleShare File Server application provides part of the user interface for the File Server Extension. Users start the file server by opening the AppleShare File Server application. The AppleShare File Server application also provides the interface for controlling and monitoring the file server while it is running. The AppleShare File Server application displays the Volume Info window, which lists the volumes that are available on the server, and the Connected Users window, which lists users who are currently logged on to the server. The Server menu lets you unmount volumes, disconnect users, send messages to users, and set the greeting message. (See the AppleShare Server 3.0 Administrator's Guide for more information about the features of the AppleShare File Server 3.0 user interface.)

The AppleShare Installer initially installs the AppleShare File Server application in the System Folder, but the file can reside anywhere on the server volume. The AppleShare File Server application communicates with the File Server Extension primarily by means of server control calls.

AppleShare Admin The AppleShare Admin application provides the user interface for defining users and groups for the server. The AppleShare Admin application also lets you set preferences, set access privileges, and perform other administrative tasks for the file server. (See the AppleShare Server 3.0 Administrator's Guide for more information about the administrative features of the AppleShare File Server 3.0.)

Like the AppleShare File Server application, the AppleShare Admin application is initially installed in the System Folder but can reside anywhere on the server volume. It communicates with the File Server Extension primarily by means of server control calls. It also uses the AppleShare PDS file and the Users & Groups Data File to store and retrieve information about server volumes and the users and groups defined for the server, respectively.

Network AppleShare clients Network workstations with AppleShare client software installed can connect to the File Server Extension. AppleShare clients communicate with the server through AFP sessions.

File Manager The Macintosh File Manager normally handles local requests for file access. While the file server is running, however, the File Server Extension intercepts all file access calls from the File Manager.

Server additions Applications, INITs, extensions, and other types of programs can access the File Server Extension by using server control calls. A program that uses server control calls is referred to as a server addition. This guide tells you how to create server additions by using server control calls in your own programs.

Macintosh File Sharing software components

Like the AppleShare File Server 3.0, Macintosh File Sharing is composed of a number of parts distributed across several files in the System Folder. The File Sharing Extension provides the actual functionality of the AFP server. Five other files -- the Network Extension, three control panels, and the Finder -- work together to provide the user interface.

The File Sharing Extension handles all requests for access to files residing on local volumes, including local requests from the Macintosh File Manager and server additions, and remote requests from AFP clients.

This section describes the software components of Macintosh File Sharing. The section "Data Files," later in this chapter, describes the Users & Groups Data File and the AppleShare PDS file.

File Sharing Extension The File Sharing Extension contains the actual file server code.

It is a system extension that resides in the Extensions folder. The File Sharing Extension is a launchable file, though its file type is 'INIT' instead of 'APPL', which prevents users from starting it from the Finder. (The 'INIT' file type also tells the system to put the file in the Extensions folder and causes the extension to be opened during system startup.) When the File Sharing Extension is launched, it runs as a background application.

The File Sharing Extension contains no user interface of its own. The user interface is provided by the Network Extension, which allows users to start and to control the File Sharing Extension. The File Sharing Extension communicates with the Network Extension primarily by means of server control calls. The File Sharing Extension communicates with the Finder by means of PPC events, and with a remote AppleShare client through AFP sessions. The File Sharing Extension also communicates with local volumes and files by means of Macintosh File Manager routines, and with server additions by means of server control calls.

When the File Sharing Extension is launched, it checks its environment, the Users & Groups Data File, and the desktop databases and AppleShare PDS files of appropriate volumes. (The File Sharing Extension does not attempt to share remote volumes, or volumes such as floppy disks or volumes that are ejected and off line during startup.) If an important required condition is not satisfied, the offending volume will not be prepared for use with the file server or file sharing will not be enabled.

Once started, the File Sharing Extension takes over the dispatching of all file system calls -- both local calls and remote requests. Essentially, the File Sharing Extension acts as a mediator between the network and your local HFS volumes. The File Sharing Extension imposes access privilege constraints on AFP requests and implements some calls that are not implemented in HFS -- such as those that govern byte-range locking, access privileges, and extended file access permissions.

Network Extension The Network Extension provides the user interface for Macintosh File Sharing. It is an extension of the Finder and resides in the Extensions folder. The Network Extension is dynamically linked with the Finder code at startup time and uses the Finder's code to control its user interface. The user interface includes what appears to users to be the Sharing Setup, Users & Groups, and File Sharing Monitor control panels. When a user opens any one of these control panels, the Network Extension code intercepts the launch command, opens the appropriate window, and controls the interaction with the user.

Based on user interactions, the Network Extension communicates with the server primarily by means of server control calls. The File Server Extension communicates with users through the Network Extension by sending high-level Apple events to display dialog boxes. The Network Extension relies on the AppleShare PDS file and the Users & Groups Data File for information about server volumes and the users and groups defined for the server, respectively.

Finder The Finder provides part of the Macintosh File Sharing services. The Sharing menu item in the Finder's File menu lets users view and set the access privileges for disks and folders. In addition, through its extension mechanism, the Finder provides an environment for running the Network Extension code. The Finder communicates with the file server by using augmented Macintosh File Manager routines.

File Sharing Monitor, Sharing Setup, and Users & Groups These control panel files trigger execution of the appropriate Network Extension code. The control panel files themselves contain no executable code.

Network AppleShare clients Network workstations with AppleShare client software installed can connect to the File Sharing Extension. AppleShare clients communicate with the server by means of AFP packets.

File Manager The Macintosh File Manager normally handles local requests for file access. When Macintosh File Sharing is turned on, however, the File Sharing Extension intercepts all file access calls from the File Manager.

Server additions Applications, INITs, extensions, and other types of programs can access the File Server Extension by using server control calls. A program that uses server control calls is referred to as a server addition. This guide tells you how to create server additions by using server control calls in your own programs.

Data files

Both the AppleShare File Server 3.0 and Macintosh File Sharing use two data files to store user and directory information: the Users & Groups Data File and the AppleShare PDS file.

Users & Groups Data File The Users & Groups Data File contains a database of the users and groups defined on your computer. You define users and groups for the AppleShare File Server 3.0 by using the AppleShare Admin application. You define users and groups with Macintosh File Sharing by using the Users & Groups control panel. The data file is a B-Tree file. With the AppleShare File Server 3.0, the AppleShare Admin and File Server Extension files use the Users & Groups Data File. With Macintosh File Sharing, the Network Extension and File Sharing Extension files use the Users & Groups Data File.

AppleShare PDS The AppleShare PDS file is an invisible file that resides at the root of every unlocked volume. PDS stands for parallel directory structure. The AppleShare PDS file contains the access privilege and share-point information for the volume on which the file resides. The PDS file determines the access privileges of the volume's users and groups, which are defined in the Users & Groups Data File. Because the PDS file is created in conjunction with the Users & Groups Data File, the Users & Groups Data File must not be removed from the volume. (If the Users & Groups Data File is lost, the access privilege and share-point information contained in the PDS file is lost as well.)

The PDS file for CD-ROM drives resides in the File Sharing folder (in the Preferences folder) for Macintosh File Sharing, and in the File Server folder (in the Preferences folder) for the AppleShare File Server 3.0.

Using server control calls

This section presents each of the server control calls available with the AppleShare File Server 3.0. The server control calls are presented in logical functional groups, more or less in the order in which you would use them in server additions.

For each server control call, there is a function named `MycallName`. Typically, this function shows how to fill in the parameter block, make the server control call, and retrieve any result values returned by the call. When necessary, the `MycallName` functions also include code that makes the server control calls supported by Macintosh File Sharing behave as much as possible in the same manner as they behave when used under the AppleShare File Server 3.0. For example, a `MycallName` function may return a value that is normally returned by the AppleShare File Server 3.0 but that is not supplied by Macintosh File Sharing. In other cases, for example with the `SCGetExpFldr` call, the function contains code that makes the call work "as advertised."

Some of the server control call descriptions are accompanied by a second segment of sample code that shows a particular use of that call. Each of these additional code segments performs one of the following tasks:

- gets the server version, status, and setup information and then stores that information in global variables for later use by other functions
- lists the shared volumes and folders
- lists the installed server event handlers
- lists the users connected to the server
- gets the mount information for a user's mounted volumes or folders
- disconnects a user from the server
- sends a message to all connected users
- disconnects the users of a specified volume

Some of the code samples use global variables to store information that might be needed by other functions. Here are the global variables used within these code samples:

- General
- gErr: OSErr
- gHasServerDispatch: Boolean
- Used by SCServerVersion
- gServerExtensionName: Str31
- gServerType: Integer
- gServerVersion: Integer
- Used by SCPollServer
- gServerState: Integer;
- gDisconnectState: Integer;
- gServerError: Integer;
- gSecondsLeft: LongInt;
- Used by SCGetSetupInfo
- gSetupInfoRec: SetupInfoRec;
- gMaxVolumes: Integer
- gMaxExpFolders: Integer
- gCurMaxSessions: Integer

Determining if server control calls are available

Before using any of the other control calls, use the `TrapAvailable` call to make sure that the server dispatch trap is available. The following line of code tests directly for the existence of the server dispatch trap:

```
gHasServerDispatch := TrapAvailable(ServerDispatch)
```

The "Compatibility Guidelines" chapter of *Inside Macintosh*, Volume VI, contains the source code for the `TrapAvailable` call.

Calling conventions

After assuring that server control calls are available, issue the `SyncServerDispatch` call with the following code:

```
scErr:=SyncServerDispatch(@scPB);
```

The actual interface for the `SyncServerDispatch` call is defined by the server control call interface file. See Appendix B for a listing of that file. The `SyncServerDispatch` call appears in the "Server Control Routine" section of Appendix B.

Getting and modifying server

configuration information

This section describes the server control calls that you use to get and to modify server configuration information.

`SCServerVersion`

The following function calls `SCServerVersion` to get the name of the file server extension and the server's type and version.

```
FUNCTION MySCServerVersion (ExtNamePtr: StringPtr;  
VAR ServerType: Integer;  
VAR ServerVersion: Integer):  
OSErr;  
VAR  
scPB: SCParamBlockRec;  
BEGIN  
scPB.versionPB.scCode := SCServerVersion;  
scPB.versionPB.scExtNamePtr := ExtNamePtr;  
MySCServerVersion := SyncServerDispatch(@scPB);  
ServerType := scPB.versionPB.scServerType;  
ServerVersion := scPB.versionPB.scServerVersion;
```

END;

The following segment of code gets the server version information and stores it in global variables for later use. (Global variables and their data types are listed in "Using Server Control Calls," earlier in this chapter.)

```
err := MySCServerVersion(@gServerExtensionName,  
gServerType, gServerVersion);
```

SCGetSetupInfo

The following function calls SCGetSetupInfo to get the file server's setup information. If the server is a Macintosh File Sharing server (type = MFSType), this function also fills in the fields that aren't returned by the server control call. Before using this function, you must initialize gServerType by using the SCServerVersion control call.

```
FUNCTION MySCGetSetupInfo (SetupPtr: SetupInfoRecPtr;
```

```
VAR MaxVolumes: Integer;
```

```
VAR MaxExpFolders: Integer;
```

```
VAR CurMaxSessions: Integer):
```

```
OSErr;
```

```
VAR
```

```
scPB: SCParamBlockRec;
```

```
BEGIN
```

```
scPB.setupPB.scCode := SCGetSetupInfo;
```

```
scPB.setupPB.scSetupPtr := SetupPtr;
```

```
MySCGetSetupInfo := SyncServerDispatch(@scPB);
```

```
CASE gServerType OF
```

```
MFSType:
```

```
BEGIN
```

```
MaxVolumes := 10;
```

```
MaxExpFolders := 10;
```

```
CurMaxSessions := SetupPtr^.SISMaxLogins;
```

```
END;
```

```
OTHERWISE
```

```
BEGIN
```

```

MaxVolumes := scPB.setupPB.scMaxVolumes;
MaxExpFolders := scPB.setupPB.scMaxExpFolders;
CurMaxSessions := scPB.setupPB.scCurMaxSessions;
END;
END;
END;

```

The following segment of code gets the server version information and stores it in global variables for later use. (Global variables and their data types are listed in "Using Server Control Calls," earlier in this chapter.)

```

err := MySCGetSetupInfo(@gSetupInfoRec, gMaxVolumes,
gMaxExpFolders, gCurMaxSessions);
SCSetSetupInfo

```

The following function calls SCSetSetupInfo to set the file server's setup information.

```

FUNCTION MySCSetSetupInfo (SetupPtr: SetupInfoRecPtr):

```

```

OSError;
VAR
scPB: SCPParamBlockRec;
BEGIN
scPB.setupPB.scCode := SCSetSetupInfo;
scPB.setupPB.scSetupPtr := SetupPtr;
MySCSetSetupInfo := SyncServerDispatch(@scPB);
END;

```

Checking the server's status

This section describes the server control calls that you use to check the server's status.

SCPollServer

The SCPollServer call returns information about the server's state, its disconnect state, whether or not there has been an error, and how many seconds are left before the server shuts down or before it disconnects a user. The following function calls SCPollServer to get this information.

```

FUNCTION MySCPollServer (VAR ServerState: Integer;
VAR DisconnectState: Integer;

```

```

VAR ServerError: Integer;
VAR SecondsLeft: LongInt): OSErr;

VAR
scPB: SCParamBlockRec;

BEGIN

scPB.pollServerPB.scCode := SCPollServer;
{ Macintosh File Sharing doesn't return scSecondsLeft }
{ so zero it. }
scPB.pollServerPB.scSecondsLeft := 0;
MySCPollServer := SyncServerDispatch(@scPB);
ServerState := scPB.pollServerPB.scServerState;
DisconnectState := scPB.pollServerPB.scDisconnectState;
ServerError := scPB.pollServerPB.scServerError;
SecondsLeft := scPB.pollServerPB.scSecondsLeft;

END;

```

The following segment of code gets the server state, disconnect state, server error, and seconds-left information and stores it in global variables for later use. (Global variables and their data types are listed in "Using Server Control Calls," earlier in this chapter.)

```

err := MySCPollServer(gServerState, gDisconnectState,
gServerError, gSecondsLeft);

```

SCGetServerStatus

The following function calls SCGetServerStatus to get the file server's current status information, including the server flags, the number of active sessions, the date of the last modification of the user list, the level of server activity, and the date of the last modification of the volume list.

Note This call is not supported by Macintosh File Sharing.

```

FUNCTION MySCGetServerStatus (NamePtr: StringPtr;

```

```

VAR ServerFlags: Integer;
VAR NumSessions: Integer;
VAR UserListModDate: LongInt;
VAR Activity: Integer;

```

```

VAR VollListModDate: LongInt);
OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.statusPB.scCode := SCGetServerStatus;
scPB.statusPB.scNamePtr := NamePtr;
MySCGetServerStatus := SyncServerDispatch(@scPB);
ServerFlags := scPB.statusPB.scServerFlags;
NumSessions := scPB.statusPB.scNumSessions;
UserListModDate := scPB.statusPB.scUserListModDate;
Activity := scPB.statusPB.scActivity;
VollListModDate := scPB.statusPB.scVollListModDate;
END;

```

Starting and stopping the file service

This section describes the server control calls that you use to start and stop file servers.

SCStartServer

The following function calls SCStartServer to start the Macintosh File Sharing server.

!! IMPORTANT The AppleShare File Server 3.0 is normally started by the AppleShare File Server application. When the AppleShare File Server application is launched, it checks to see if the file server is running. If it is, the AppleShare File Server application assumes its role as the file server's user interface. If the file server is not running, the AppleShare File Server application starts the server by calling SCStartServer before assuming its role as user interface.

If a server addition starts the AppleShare File Server 3.0 by calling SCStartServer, the file service starts up, but the AppleShare File Server application (the user interface) does not. Unless your program provides the functionality of the AppleShare File Server application, it should probably not call SCStartServer to start the AppleShare File Server 3.0. Instead, start the file server in the usual way -- by launching the AppleShare File Server application. !!

```

FUNCTION MySCStartServer: OSErr;

```

```

VAR

```

```
scPB: SCParamBlockRec;
BEGIN
scPB.startPB.scCode := SCStartServer;
scPB.startPB.scStartSelect := kCurInstalled;
scPB.startPB.scEventSelect := kFinderExtn;
MySCStartServer := SyncServerDispatch(@scPB);
END;
```

SCShutDown

The following function calls SCShutDown to shut down the file server.

!! IMPORTANT The AppleShare File Server application automatically quits if the AppleShare File Server 3.0 is shut down with the SCShutDown call. !!

```
FUNCTION MySCShutDown (NumMinutes: Integer; Flags: Integer;
MessagePtr: StringPtr): OSErr;
```

VAR

```
scPB: SCParamBlockRec;
BEGIN
scPB.disconnectPB.scCode := SCShutDown;
scPB.disconnectPB.scNumMinutes := NumMinutes;
scPB.disconnectPB.scFlags := Flags;
scPB.disconnectPB.scMessagePtr := MessagePtr;
MySCShutDown := SyncServerDispatch(@scPB);
END;
```

SCCancelShutDown

The following function calls SCCancelShutDown to cancel a shutdown or disconnect call in progress.

```
FUNCTION MySCCancelShutDown: OSErr;
```

VAR

```
scPB: SCParamBlockRec;
BEGIN
scPB.disconnectPB.scCode := SCCancelShutDown;
```

```
MySCCancelShutDown := SyncServerDispatch(@scPB);
```

```
END;
```

```
SCSleepServer
```

The following function calls SCSleepServer to temporarily shut down the file server ("put it to sleep"). You might want to put a file server to sleep before switching networks or temporarily turning off AppleTalk.

- Note This call is not supported by Macintosh File Sharing.

```
FUNCTION MySCSleepServer (NumMinutes: Integer;
```

```
Flags: Integer;
```

```
MessagePtr: StringPtr): OSErr;
```

```
VAR
```

```
scPB: SCParamBlockRec;
```

```
BEGIN
```

```
scPB.disconnectPB.scCode := SCSleepServer;
```

```
scPB.disconnectPB.scNumMinutes := NumMinutes;
```

```
scPB.disconnectPB.scFlags := Flags;
```

```
scPB.disconnectPB.scMessagePtr := MessagePtr;
```

```
MySCSleepServer := SyncServerDispatch(@scPB);
```

```
END;
```

```
SCWakeServer
```

The following function calls SCWakeServer to reactivate an AppleShare File Server 3.0 that has been temporarily shut down (that is, a file server that is "sleeping").

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION MySCWakeServer: OSErr;
```

```
VAR
```

```
scPB: SCParamBlockRec;
```

```
BEGIN
```

```
scPB.startPB.scCode := SCWakeServer;
```

```
MySCWakeServer := SyncServerDispatch(@scPB);
```

```
END;
```

Obtaining status information about users, volumes, and shared items

This section describes the server control calls that you use to obtain information about file server users, volumes, and shared volumes and folders.

SCGetExpFldr

The following function calls SCGetExpFldr to get information from the call's return parameters about shared volumes and folders at a specified index position. The return parameters provide information such as a folder's AFP short name and directory ID, the number of users who have mounted the volume or folder, and the index of a volume or folder. (See "SCGetExpFldr" in Chapter 3 for detailed descriptions of the call's return parameters.) Before using this function, you must initialize gServerType with the value returned by the SCServerVersion control call.

```
FUNCTION MySCGetExpFldr (NamePtr: StringPtr;
VAR VRefNum: Integer;
VAR Logins: Integer;
Index: Integer;
VAR DirID: LongInt): OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.standardPB.scCode := SCGetExpFldr;
{ Initialize scVRefNum to 0 so we can tell if }
{ SCGetExpFldr returned something when used with }
{ Macintosh File Sharing }
scPB.standardPB.scVRefNum := 0;
IF Index < 0 THEN
BEGIN
{ File Sharing trashes memory if (scIndex < 0) and }
{ (scNamePtr <> NIL), so we'll prevent that from }
{ happening here. }
scPB.standardPB.scNamePtr := NIL;
{ and we'll return an empty string }
IF NamePtr <> NIL THEN
```

```

NamePtr^ := '';
END
ELSE
BEGIN
scPB.standardPB.scNamePtr := NamePtr;
END;
scPB.standardPB.scIndex := Index;
MySCGetExpFldr := SyncServerDispatch(@scPB);
CASE gServerType OF
MFSType:
BEGIN
IF scPB.standardPB.scVRefNum <> 0 THEN
BEGIN
VRefNum := scPB.standardPB.scVRefNum;
Logins := 0;
DirID := scPB.standardPB.scDirID;
END
ELSE { there's nothing at this index position }
{ so force the error code to make it act }
{ like AppleShare }
MySCGetExpFldr := fnfErr;
END;
OTHERWISE
BEGIN
VRefNum := scPB.standardPB.scVRefNum;
Logins := scPB.standardPB.scLogins;
DirID := scPB.standardPB.scDirID;
END;
END;

```

END;

SERVER CONTROL CALLS

The following procedure creates a list of shared volumes and folders. Before using this procedure, you must initialize `gMaxVolumes` and `gMaxExpFolders` with the values returned by the `SCGetGetupInfo` control call.

PROCEDURE GetAllExpFldrs;

VAR

Index: Integer;

shortName: Str13;

VRefNum: Integer;

Logins: Integer;

DirID: LongInt;

err: OSErr;

BEGIN

FOR Index := -gMaxVolumes TO gMaxExpFolders DO

IF Index <> 0 THEN { index 0 is undefined }

BEGIN

err := MySCGetExpFldr(@shortName, VRefNum, Logins,

Index, DirID);

IF err = noErr THEN

BEGIN

IF Index < 0 THEN

BEGIN

{ do something with the shared volume }

{ information }

END

ELSE

BEGIN

{ do something with the shared folder }

{ information }

```

END;
END
ELSE IF err <> fnfErr THEN
{ fnfErr only means there is nothing at this }
{ Index position }
BEGIN
{ handle any unexpected errors }
END;
END;
END;

```

SCGetUserNameRec

The following function calls SCGetUserNameRec to get information about a user connected to the file server.

Note This call is not supported by Macintosh File Sharing.

```

FUNCTION MySCGetUserNameRec (NamePtr: StringPtr;
VAR Position: LongInt;
VAR UNRecID: LongInt;
VAR UserID: LongInt;
VAR LoginTime: LongInt;
VAR LastUseTime: LongInt;
VAR SocketNum: AddrBlock):
OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.userInfoPB.scCode := SCGetUserNameRec;
scPB.userInfoPB.scNamePtr := NamePtr;
scPB.userInfoPB.scPosition := Position;
MySCGetUserNameRec := SyncServerDispatch(@scPB);
Position := scPB.userInfoPB.scPosition;

```

```
UNRecID := scPB.userInfoPB.scUNRecID;
UserID := scPB.userInfoPB.scUserID;
LoginTime := scPB.userInfoPB.scLoginTime;
LastUseTime := scPB.userInfoPB.scLastUseTime;
SocketNum := scPB.userInfoPB.scSocketNum;
END;
```

The following procedure creates a list of the users logged on to a file server.

```
PROCEDURE GetAllUserNameRecs;
VAR
  err: OSErr;
  UserName: Str31;
  Position: LongInt;
  UNRecID: LongInt;
  UserID: LongInt;
  LoginTime: LongInt;
  LastUseTime: LongInt;
  SocketNum: AddrBlock;
BEGIN
  Position := 0;
  REPEAT
    err := MySCGetUserNameRec(@UserName, Position, UNRecID,
  UserID, LoginTime, LastUseTime, SocketNum);
    IF err = noErr THEN
      BEGIN
        { do something with the user information returned }
      END
    ELSE IF err <> fnfErr THEN
      { fnfErr only means there are no more users }
    BEGIN
```

```
{ handle any unexpected errors }
```

```
END;
```

```
UNTIL err <> noErr;
```

```
END;
```

```
SCGetUserMountInfo
```

The following function calls SCGetUserMountInfo to get information about the status of a particular volume or shared folder, such as the number of open files on the volume, the number of files that are open with write access, whether the volume is mounted, and whether the volume is mounted with owner privileges (that is, whether the user is a superuser).

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION MySCGetUserMountInfo (VRefNum: Integer;
```

```
VAR FilesOpen: Integer;
```

```
VAR WriteableFiles: Integer;
```

```
UNRecID: LongInt;
```

```
VAR Mounted: Boolean;
```

```
VAR MountedAsOwner: Boolean):
```

```
OSErr;
```

```
VAR
```

```
scPB: SCParamBlockRec;
```

```
BEGIN
```

```
scPB.volMountedPB.scCode := SCGetUserMountInfo;
```

```
scPB.volMountedPB.scVRefNum := VRefNum;
```

```
scPB.volMountedPB.scUNRecID := UNRecID;
```

```
MySCGetUserMountInfo := SyncServerDispatch(@scPB);
```

```
FilesOpen := scPB.volMountedPB.scFilesOpen;
```

```
WriteableFiles := scPB.volMountedPB.scWriteableFiles;
```

```
Mounted := scPB.volMountedPB.scMounted;
```

```
MountedAsOwner := scPB.volMountedPB.scMountedAsOwner;
```

```
END;
```

The following procedure gets the user-mount information for all of the volumes and shared folders that a user has mounted. Before using this

procedure, you must

initialize `gMaxVolumes` and `gMaxExpFolders` with the values returned by the `SCGetGetupInfo` control call.

```
PROCEDURE GetAllUserMountInfo (UNRecID: LongInt);
```

```
VAR
```

```
err: OSErr;
```

```
Index: Integer;
```

```
VRefNum: Integer;
```

```
FilesOpen: Integer;
```

```
WritableFiles: Integer;
```

```
Mounted: Boolean;
```

```
MountedAsOwner: Boolean;
```

```
BEGIN
```

```
FOR Index := -gMaxVolumes TO gMaxExpFolders DO
```

```
IF Index <> 0 THEN { index 0 is undefined }
```

```
BEGIN
```

```
err := MySCGetUserMountInfo(VRefNum, FilesOpen,
```

```
WritableFiles, UNRecID,
```

```
Mounted, MountedAsOwner);
```

```
IF (err = noErr) AND Mounted THEN
```

```
BEGIN
```

```
{ do something with the information returned }
```

```
END;
```

```
END;
```

```
END;
```

Disconnecting users

This section describes the server control calls that you use to disconnect users from file servers and from file server volumes.

`SCDisconnect`

The following function calls `SCDisconnect` to disconnect specified users from

a file server.

Note Although Macintosh File Sharing implements SCDisconnect, there is no way to use this call with Macintosh File Sharing because Macintosh File Sharing does not implement the SCGetUserNameRec call. SCGetUserNameRec retrieves information -- namely user name record IDs (UNRecID) -- that is necessary for SCDisconnect to work.

```
FUNCTION MySCDisconnect (DiscArrayPtr: LongIntPtr;
```

```
ArrayCount: Integer;
```

```
NumMinutes: Integer;
```

```
Flags: Integer;
```

```
MessagePtr: StringPtr): OSErr;
```

```
VAR
```

```
scPB: SCParamBlockRec;
```

```
BEGIN
```

```
scPB.disconnectPB.scDiscArrayPtr := DiscArrayPtr;
```

```
scPB.disconnectPB.scArrayCount := ArrayCount;
```

```
scPB.disconnectPB.scCode := SCDisconnect;
```

```
scPB.disconnectPB.scNumMinutes := NumMinutes;
```

```
scPB.disconnectPB.scFlags := Flags;
```

```
scPB.disconnectPB.scMessagePtr := MessagePtr;
```

```
MySCDisconnect := SyncServerDispatch(@scPB);
```

```
END;
```

The following procedure delivers a disconnect message to and disconnects the specified user after ten minutes.

```
PROCEDURE DisconnectUser (UNRecID: LongInt);
```

```
VAR
```

```
err: OSErr;
```

```
ArrayCount: Integer;
```

```
NumMinutes: Integer;
```

```
Flags: Integer;
```

```
Message: tLoginMsg;
```

```
BEGIN
```

```

ArrayCount := 1; { one user }
NumMinutes := 10;
Flags := UNRFSendMsgMask; { send a message }
Message := 'Goodbye.';
err := MySCDisconnect(@UNRecID, ArrayCount, NumMinutes,
Flags, @Message);
IF err = noErr THEN
{ the disconnect was started }
ELSE
BEGIN
{ handle any errors }
END;
END;

```

SCDisconnectVolUsers

The following function calls SCDisconnectVolUsers to disconnect the users of specified volumes.

Note This call is not supported by Macintosh File Sharing.

```

FUNCTION MySCDisconnectVolUsers (DiscArrayPtr: LongIntPtr;
ArrayCount: Integer;
NumMinutes: Integer;
Flags: Integer;
MessagePtr: StringPtr):
OSErr;
VAR
scPB: SCPParamBlockRec;
BEGIN
scPB.disconnectPB.scDiscArrayPtr := DiscArrayPtr;
scPB.disconnectPB.scArrayCount := ArrayCount;
scPB.disconnectPB.scCode := SCDisconnectVolUsers;
scPB.disconnectPB.scNumMinutes := NumMinutes;

```

```
scPB.disconnectPB.scFlags := Flags;
scPB.disconnectPB.scMessagePtr := MessagePtr;
MySCDisconnectVolUsers := SyncServerDispatch(@scPB);
END;
```

The following procedure delivers a message to and disconnects the users of the specified volume after ten minutes.

```
PROCEDURE DisconnectVolUsers (VRefNum: Integer);
VAR
err: OSErr;
VolToDisconnect: LongInt;
ArrayCount: Integer;
NumMinutes: Integer;
Flags: Integer;
Message: tLoginMsg;
BEGIN
VolToDisconnect := VRefNum; { note: Integer -> LongInt }
ArrayCount := 1;
NumMinutes := 10;
Flags := UNRFSendMsgMask; { send a message }
Message := 'A volume is going away.';
err := MySCDisconnectVolUsers(@VolToDisconnect,
ArrayCount, NumMinutes, Flags, @Message);
IF err = noErr THEN
{ the disconnect was started }
ELSE
BEGIN
{ handle any errors }
END;
END;
Sending messages to users
```

This section describes the server control call that lets you send messages to users.

SCSendMessage

The following function calls SCSendMessage to send a message to the users specified in the array pointed to by DiscArrayPtr.

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION MySCSendMessage (DiscArrayPtr: LongIntPtr;
ArrayCount: Integer;
Flags: Integer;
MessagePtr: StringPtr): OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.disconnectPB.scDiscArrayPtr := DiscArrayPtr;
scPB.disconnectPB.scArrayCount := ArrayCount;
scPB.disconnectPB.scCode := SCSendMessage;
scPB.disconnectPB.scFlags := Flags;
scPB.disconnectPB.scMessagePtr := MessagePtr;
MySCSendMessage := SyncServerDispatch(@scPB);
END;
```

The following procedure sends a message to all connected users. Before using this routine, you must initialize gCurMaxSession by using the SCGetSetupInfo call.

```
PROCEDURE SendMessageToAll;
{ This routine depends on gCurMaxSessions being }
{ initialized with SCGetSetupInfo. }
VAR
err: OSErr;
ArrayPosPtr: LongIntPtr;
Position: LongInt;
scPB: SCParamBlockRec;
```

```

DiscArrayPtr: LongIntPtr;
ArrayCount: Integer;
Flags: Integer;
Message: tLoginMsg;
BEGIN
{ allocate an array large enough to get all users }
DiscArrayPtr :=
LongIntPtr(NewPtr(sizeof(LongInt) * gCurMaxSessions));
IF DiscArrayPtr <> NIL THEN
BEGIN
Position := 0;
ArrayCount := 0;
ArrayPosPtr := DiscArrayPtr;
REPEAT
{ build list of users with SCGetUserNameRec }
scPB.userInfoPB.scCode := SCGetUserNameRec;
scPB.userInfoPB.scNamePtr := NIL;
scPB.userInfoPB.scPosition := Position;
err := SyncServerDispatch(@scPB);
IF err = noErr THEN
BEGIN { add user to array }
ArrayPosPtr^ := scPB.userInfoPB.scUNRecID;
ArrayPosPtr := LongIntPtr(ORD4(ArrayPosPtr) +
sizeof(LongInt));
ArrayCount := ArrayCount + 1;
END;
UNTIL err <> noErr;
IF ArrayCount > 0 THEN
BEGIN

```

```

Flags := UNRFSendMsgMask; { send a message }
Message := 'Moof!';
err := MySCSendMessage(DiscArrayPtr, ArrayCount,
Flags, @Message);
IF err = noErr THEN
{ the message was sent }
ELSE
BEGIN
{ handle any errors from SCSendMessage }
END
END
ELSE { there are no users connected }
; { do nothing }
DisposPtr(Ptr(DiscArrayPtr));
END
ELSE
BEGIN
{ handle memory manager error }
END;
END;

```

Setting or clearing the copy-protect status of files

This section describes the server control calls that let you set or clear the copy-protect status of files on a file server.

SCSetCopyProtect

The following function calls SCSetCopyProtect to set the copy-protect status of a file.

Note This call is not supported by Macintosh File Sharing.

```

FUNCTION MySCSetCopyProtect (NamePtr: StringPtr;
VRefNum: Integer;
DirID: LongInt): OSErr;

```

VAR

scPB: SCParamBlockRec;

BEGIN

scPB.standardPB.scNamePtr := NamePtr;

scPB.standardPB.scVRefNum := VRefNum;

scPB.standardPB.scCode := SCSetCopyProtect;

scPB.standardPB.scDirID := DirID;

MySCSetCopyProtect := SyncServerDispatch(@scPB);

END;

SCClrCopyProtect

The following function calls SCClrCoyProtect to clear the copy-protect status of a file.

Note This call is not supported by Macintosh File Sharing.

FUNCTION MySCClrCopyProtect (NamePtr: StringPtr;

VRefNum: Integer;

DirID: LongInt): OSErr;

VAR

scPB: SCParamBlockRec;

BEGIN

scPB.standardPB.scNamePtr := NamePtr;

scPB.standardPB.scVRefNum := VRefNum;

scPB.standardPB.scCode := SCClrCopyProtect;

scPB.standardPB.scDirID := DirID;

MySCClrCopyProtect := SyncServerDispatch(@scPB);

END;

Using server event handlers

This section describes the server control calls that you use with server event handlers. Server event handlers are discussed in Chapter 2.

SCInstallServerEventProc

The following function calls SCInstallServerEventProc to install a server event handler.

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION mySCInstallServerEventProc
(theSEHandler: ProcPtr): OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.serverEventPB.scSEQEntryPtr := theSEHandler;
scPB.serverEventPB.scCode := SCInstallServerEventProc;
mySCInstallServerEventProc := SyncServerDispatch(@scPB);
END;
SCRemoveServerEventProc
```

The following function calls SCRemoveServerEventProc to remove a server event handler.

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION mySCRemoveServerEventProc (theSEHandler: ProcPtr):
OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.serverEventPB.scSEQEntryPtr := theSEHandler;
scPB.serverEventPB.scCode := SCRemoveServerEventProc;
mySCRemoveServerEventProc := SyncServerDispatch(@scPB);
END;
SCGetServerEventProc
```

The following function calls SCGetServerEventProc to get a pointer to the head of the server event handler queue.

Note This call is not supported by Macintosh File Sharing.

```
FUNCTION MySCGetServerEventProc (VAR theSEQHdrPtr:
QHdrPtr): OSErr;
VAR
```

```
scPB: SCParamBlockRec;

BEGIN

scPB.serverEventPB.scCode := SCGetServerEventProc;

MySCGetServerEventProc := SyncServerDispatch(@scPB);

theSEQHdrPtr := QHdrPtr(scPB.serverEventPB.scSEQEntryPtr);

END;
```

CHAPTER 2

SERVER EVENT HANDLING

The chapter explains how your applications can monitor server events and respond to these events by using server event handlers. A sample handler is included to show how you might implement server event handlers in your own server additions.

The AppleShare File Server 3.0 server event mechanism enables programs (and INITs) to monitor and respond to a file server's activities. This mechanism allows developers to create programs that work in concert with file servers to extend the services provided by the servers. For example, server statistics reporting, audit trailing, and extended security could all be added to existing file services.

The server event mechanism comprises two parts: the server event handler and the application program. The server event handler is a server-addition procedure, installed in the server by the SCInstallServerEventProc server control call. The server calls the server event handler whenever a server event occurs. A server event is a condition or operation occurring in the file server, such as the receipt of an AFP or server control call, the mounting of a volume by a user, or a client disconnect. When a server notifies the server event handler of an event, the handler passes information to the application program so that the program can respond to the event. An application typically allocates a buffer and passes the buffer's address to the server event handler when the handler is installed. The server event handler fills the buffer asynchronously, while the installing program analyzes the buffer's contents from the application's event loop.

Using server events

To monitor server events from your server addition, you must first install a server event handler in the file server. You install a server event handler from your program by issuing the SCInstallServerEventProc server control call, as described in "Using Server Event Handlers" in Chapter 1. Installing a server event handler is very similar to the process of installing the AppleTalk Transition Queue. (For information about installing an entry into the AppleTalk Transition Queue, see *Inside Macintosh*, Volume VI.)

Once the server event handler is installed, it gains control whenever one of the specified server events occurs. When a server event occurs, the server determines whether any server event handlers are installed. For each installed handler, the server checks the SEeventFlag in the tSEQEntry record

to see if the handler is interested in the event that just happened. If it is, the server calls the handler, passing pointers to the tSEQEntry record and a server event record owned by the server. It is up to the event handler to copy the server event record into the application's own buffer.

The server event record contains, among other things, the following information:

- which server event occurred
- the time of the server event (in standard Macintosh date/time form)
- any error associated with the call
- the size of the data in the next buffer
- a buffer containing the AFP packet, SCPParamBlockRec, HParamBlockRec, or the new server name (up to a maximum of 48 bytes)
- the name of the file or directory upon which the operation is being performed

(if applicable)

- the AFP command
- the user's unique user name record ID (UNRecID)
- the user's user ID (SUserID)
- the user name of the user performing this operation (registered users only)
- the reference number and directory ID of the volume upon which this operation was performed (if applicable)
- the socket address of this user (provided in the address block (AddrBlock) format: net number:node ID:socket number)

The server event interface file, listed in Appendix B, contains detailed comments about each field of the server event record.

Constraints

This section describes constraints that you must observe for the server event mechanism to work properly. It is the server event handler's responsibility to copy the desired information from the server event record into its own pre-allocated buffers. The server event handler cannot make file system or Memory Manager calls while inside its thread of control. Furthermore, because it is really part of a completion routine in the file server's code, the handler must relinquish control to the server as soon as possible. It is useful to consider that the server event handler is dynamically linked into one of the completion routines of the file server and is thus an extension to it. Therefore, it is as important to minimize the time spent in the server event handler as it is to minimize the time spent in the completion routines. Every microsecond spent in the server event handler results in a corresponding delay in the completion of file server client's call.

Although you can use server events only as notification that a condition has

been satisfied, you can use server events in conjunction with server control calls to respond to the condition. For example, you can shut the server down, disconnect a user, or send a message to any or all connected users as a response to a server event.

Sample server event handler code

This section contains sample code that implements the server event mechanism in a server addition. The sample includes all of the necessary parts; you need only plug in your specific code segments to make it work. Comments within the code explain the purpose of each part. You can copy as much of the sample code as you want to use in your own server additions.

```
{ This unit contains the server event handler, the server }
{ event record processor, and related routines. }

UNIT ServerEventHandler;

{=====}

INTERFACE

USES

AppleTalk, Processes,
{$IFC UNDEFINED THINK_Pascal}
Errors, Memory, Packages,
{$ENDC}

ServerControlIntf, ServerEventIntf;

CONST

{ This value indicates how many server events can be }
{ queued for ProcessServerEvents to handle later. }
{ If you expect a large number of server events to }
{ come in over a short amount of time, increase this }
{ number. }

kNumberServerEvents = 100;

TYPE

{ Add required queue element fields to a server event }
{ record so we can use it as an OS queue element. }

SERecQElemPtr = ^SERecQElem;

SERecQElem = RECORD
```

```

qLink: QElemPtr;
qType: Integer;
theSERec: ServerEventRecord;
END;
{ Extend the tSEQEntry with a few items we need access }
{ to within the server event handler. }
ExtendedSEQEntryPtr = ^ExtendedSEQEntry;
ExtendedSEQEntry = RECORD
theSEQEntry: tSEQEntry; { A server event queue entry. }
freeQ, usedQ: QHdr;     { Queue headers for server }
{ event record queues. }
seRecArrayPtr: Ptr;     { Pointer to allocated array }
{ of SERecQElem. }
ourPSN: ProcessSerialNumber; { The application's PSN. }
END;
VAR
{ The global extended tSEQEntry record. }
gExtendedSEQEntry: ExtendedSEQEntry;
FUNCTION InstallServerEventHandler: OSErr;
FUNCTION RemoveServerEventHandler: OSErr;
PROCEDURE ProcessServerEvents;
{=====}
IMPLEMENTATION
{ This function calls SCInstallServerEventProc to install }
{ a server event handler. }
FUNCTION mySCInstallServerEventProc
(theSEHandler: ProcPtr): OSErr;
VAR
scPB: SCParamBlockRec;

```

```

BEGIN
scPB.serverEventPB.scSEQEntryPtr := theSEHandler;
scPB.serverEventPB.scCode := SCInstallServerEventProc;
mySCInstallServerEventProc := SyncServerDispatch(@scPB);
END;

{ This function calls SCRemoveServerEventProc to remove a }
{ server event handler. }

FUNCTION mySCRemoveServerEventProc (theSEHandler: ProcPtr):
OSErr;
VAR
scPB: SCParamBlockRec;
BEGIN
scPB.serverEventPB.scSEQEntryPtr := theSEHandler;
scPB.serverEventPB.scCode := SCRemoveServerEventProc;
mySCRemoveServerEventProc := SyncServerDispatch(@scPB);
END;

{ TheSrvrEventHandler shows what should be done in a }
{ server event handler and no more: It gets a server }
{ event record from the free queue of application supplied }
{ server event records (or if the freeQ is empty, it gets }
{ the oldest server event record from the usedQ ); it }
{ copies AppleShare's server event record (pointed to by }
{ theSERecPtr) into the application's server event record; }
{ it puts the application's server event record into the }
{ used queue where it can be serviced from the }
{ application's event loop; and then calls WakeUpProcess }
{ so the event loop can handle the server event record }
{ in the queue as soon as possible. }

PROCEDURE TheSrvrEventHandler

```

```

(theSEQPtr: ExtendedSEQEntryPtr;
theSERecPtr: ServerEventRecordPtr);
VAR
theSERecQElemPtr: SERecQElemPtr;
BEGIN
WITH theSEQPtr^ DO
BEGIN
IF freeQ.qHead <> NIL THEN
BEGIN
{ Get the server event record out of the freeQ. }
theSERecQElemPtr := SERecQElemPtr(freeQ.qHead);
IF Dequeue(QElemPtr(theSERecQElemPtr), @freeQ) <>
noErr THEN
; { Do nothing with errors-- }
{ you'd better not be getting them! }
END
ELSE
BEGIN
{ The freeQ is empty, so get the oldest server }
{ event record out of the usedQ. }
theSERecQElemPtr := SERecQElemPtr(usedQ.qHead);
IF Dequeue(QElemPtr(theSERecQElemPtr), @usedQ) <>
noErr THEN
; { Do nothing with errors-- }
{ you'd better not be getting them! }
END;
{ Copy the server event record into my server }
{ event record, }
theSERecQElemPtr^.theSERec := theSERecPtr^;

```

```

{ and enqueue my server event record into }
{ the usedQ. }
Enqueue(QElemPtr(theSERecQElemPtr), @usedQ);
{ Wake up our process so it can handle the server }
{ event record ASAP. }
IF WakeUpProcess(ourPSN) <> noErr THEN
; { Do nothing with errors-- }
{ you'd better not be getting them! }
END;
END;
{ InitSEQEntry initializes the fields of gExtendedSEQEntry }
{ to zero, allocates kNumberServerEvents of SERecQElem and }
{ enqueues them into the free queue of gExtendedSEQEntry, }
{ and gets the applications process serial number and puts }
{ it in gExtendedSEQEntry so the server event handler can }
{ wake up the process. InitSEQEntry returns TRUE if the }
{ array of SERecQElem was allocated. }
FUNCTION InitSEQEntry: Boolean;
VAR
theQElemPtr: Ptr;
index: Integer;
BEGIN
WITH gExtendedSEQEntry DO
BEGIN
{ Point to server event handler. }
theSEQEntry.SEQEntry.CallAddr :=
@TheSrvrEventHandler;
{ Initially clear all SEeventFlags, }
theSEQEntry.SEeventFlag := 0;

```

```

{ clear all SEwhichAFPFlags, }
theSEQEntry.SEwhichAFPFlag[0] := 0;
theSEQEntry.SEwhichAFPFlag[1] := 0;
{ and clear all SEwhichSCFlags. }
theSEQEntry.SEwhichSCFlag := 0;
{ Allocate some memory for the server event }
{ records and initialize the buffer queues. }
seRecArrayPtr := NewPtr(kNumberServerEvents *
LongInt(sizeof(SERecQElem)));
IF seRecArrayPtr <> NIL THEN
BEGIN
{ Initialize the usedQ header. }
usedQ.qFlags := 0;
usedQ.qHead := NIL;
usedQ.qTail := NIL;
{ Initialize the freeQ header. }
freeQ.qFlags := 0;
freeQ.qHead := NIL;
freeQ.qTail := NIL;
{ The free queue holds all of our server }
{ event records initially, so add the }
{ SERecQElem to the freeQ. }
theQElemPtr := seRecArrayPtr;
FOR index := 1 TO kNumberServerEvents DO
BEGIN
Enqueue(QElemPtr(theQElemPtr), @freeQ);
theQElemPtr := Ptr(ORD4(theQElemPtr) +
LongInt(sizeof(SERecQElem)));
END;

```

```

InitSEQEntry := TRUE; { Everything is OK. }
END
ELSE
InitSEQEntry := FALSE; { No memory. }
IF GetCurrentProcess(ourPSN) <> noErr THEN
; { Get our process serial number. }
END;
END;
{ SetSEFlags sets the server event flags of }
{ gExtendedSEQEntry to tell AppleShare's server event }
{ mechanism which server events your application's server }
{ event handler are interested in. You can set the }
{ SE flags either before or after your server event }
{ handler is installed. }
{ IMPORTANT NOTES:}
{ • Your server event handler will be called based on the }
{ current settings of SEeventFlag. Make sure }
{ SEeventFlag is either initialized to zero (meaning }
{ your server event handler is not interested in any }
{ server events) or initialized for the specific server }
{ events your application is interested in before you }
{ install your server event handler. }
{ • If you set the bCSEHAFPIInDoRequest or }
{ bCSEHAFPIInSendResponse bits in SEeventFlag after your }
{ server event handler is installed, make sure you }
{ initialize the SEwhichAFPFflag bits first. }
{ • If you set the bCSEHServerControlCall bit in }
{ SEeventFlag after your server event handler is }
{ installed, make sure you initialize the SEwhichSCFlag }

```

```

{ bits first. }
PROCEDURE SetSEFlags;
BEGIN
WITH gExtendedSEQEntry.theSEQEntry DO
BEGIN
{ If the bCSEHAFPIInDoRequest or }
{ bCSEHAFPIInSendResponse bits in SEeventFlag are }
{ going to be set, then indicate what AFP calls }
{ you're interested in. For example: }
{ BSET(theSEQEntry.SEwhichAFPFlag[1], afpOpenFork); }
{ will cause a server event for the afpOpenFork }
{ AFP call. }
{ • add your code here • }
{ If the bCSEHServerControlCall bit in SEeventFlag }
{ is going to be set, }
{ then indicate what server control calls you're }
{ interested in. For example: }
{ BSET(theSEQEntry.SEwhichSCFlag, SCSetSetupInfo); }
{ will cause a server event for the SCSetSetupInfo }
{ server control call. }
{ • add your code here • }
{ Indicate what server events you'd like to be }
{ notified of by setting bits in the SEeventFlag }
{ longword. For example: }
{ BSET(SEeventFlag, bCSEHVolumePrep); }
{ will cause a server event every time the server }
{ prepares a volume for use with AppleShare. }
{ • add your code here • }
END;

```

```

END;
{ ProcessServerEvents should be called every time through }
{ the event loop to see if there are any server event }
{ records to process. If there aren't, then it exits }
{ immediately. If there are, then it processes the server }
{ event records in the used queue until none are left. }
PROCEDURE ProcessServerEvents;
VAR
theSERecQElemPtr: SERecQElemPtr;
BEGIN
WITH gExtendedSEQEntry DO
WHILE usedQ.qHead <> NIL DO
BEGIN
{ Get the server event record out of the usedQ. }
theSERecQElemPtr := SERecQElemPtr(usedQ.qHead);
IF Dequeue(QElemPtr(theSERecQElemPtr), @usedQ) =
noErr THEN
BEGIN
WITH theSERecQElemPtr^.theSERec DO
BEGIN
{ Do something useful with the }
{ server event record. }
{ • add your code here • }
END;
{ We're done with the server event }
{ record, so put it back in the freeQ. }
Enqueue(QElemPtr(theSERecQElemPtr), @freeQ);
END;
END;

```

```

END;
{ Your application calls InstallServerEventHandler to }
{ install the server event handler. Change the }
{ InitSEQEntry function where indicated to tell the server }
{ event handler mechanism what server events you're }
{ interested in to begin with. }
FUNCTION InstallServerEventHandler: OSErr;
VAR
err: OSErr;
BEGIN
{ initialize queues and get server event record buffer }
IF InitSEQEntry THEN
BEGIN
{ Set the server event flags in }
{ gExtendedSEQEntry.theSEQEntry. }
SetSEFlags;
{ Install the server event handler. }
err :=
mySCInstallServerEventProc(@gExtendedSEQEntry);
IF err <> noErr THEN { SE handler not installed? }
{ Then get rid of memory. }
DisposPtr(gExtendedSEQEntry.seRecArrayPtr);
{ Return any SCInstallServerEventProc errors. }
InstallServerEventHandler := err;
END
ELSE
{ Return a memory error. }
InstallServerEventHandler := memFullErr;
END;

```

```
{ Your application calls RemoveServerEventHandler to }
{ remove the server event handler and dispose of the }
{ memory allocated by InitSEQEntry (which is called by }
{ InstallServerEventHandler). }
```

```
FUNCTION RemoveServerEventHandler: OSErr;
```

```
BEGIN
```

```
{ Remove the server event handler. }
```

```
RemoveServerEventHandler :=
```

```
mySCRemoveServerEventProc(@gExtendedSEQEntry);
```

```
{ Get rid of memory used for the server event records. }
```

```
IF gExtendedSEQEntry.seRecArrayPtr <> NIL THEN
```

```
DisposPtr(gExtendedSEQEntry.seRecArrayPtr);
```

```
END;
```

```
END. { ServerEventHandler unit }
```

```
Application event loop
```

The heart of any Macintosh program is the event loop, which causes the application to wait for an event -- such as a user's attempt to choose a menu item or open a file. When an event occurs, the application can respond accordingly.

The following sample code shows an application event loop that processes the server events announced by a server event handler.

```
REPEAT
```

```
IF WaitNextEvent(everyEvent, event, sleep, cursorRgn) THEN
```

```
DoEvent(event);      {process toolbox events}
```

```
ProcessServerEvents;  {process server events}
```

```
UNTIL quitFlag;
```

CHAPTER 3

SERVER CONTROL CALL REFERENCE

This chapter provides detailed information about each of the AppleShare File Server 3.0 server control calls. This chapter gives a brief description of each call, shows the structure of the parameter block, describes each field

of the parameter block, and lists the possible result codes. The calls are presented in alphabetical order.

SCCancelShutDown

SCCancelShutDown cancels the shutdown or disconnect in progress. If a shutdown was in progress, a shutdown-canceled attention message is sent to all affected users.

Parameter (uses standardPB variant of SCParamBlockRec)

Block	16	ioResult	word
	26	scCode	word

Fields ioResult Word result value: Result code.

scCode Word input value: The server control code; always SCCancelShutDown (\$0003).

Result Codes noErr 0 No error.

paramErr -50 No shutdown or disconnect was in progress.

SCClrCopyProtect

SCClrCopyProtect is called by the AppleShare Admin application or some other program executing locally on the server computer when the program wants to clear the copy-protect status of a file.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses standardPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scNamePtr	long
	22	scVRefNum	word
	26	scCode	word
	30	scDirID	long

Fields ioResult Word result value: Result code.

scNamePtr Longword input pointer: Points to the filename.

scVRefNum Word input value: The volume specification.

scCode Word input value: The server control code; always SCLrCopyProtect (\$0011).

scDirID Longword input value: The parent directory ID.

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.

SCClrCopyProtect may also return errors returned by the PBGetCatInfo and PBSetCatInfo routines.

SCDisconnect

SCDisconnect disconnects every user whose user name record ID (UNRecID) is contained in the array pointed to by scDiscArrayPtr and sends a disconnect attention message to all of these users.

Note Macintosh File Sharing does not support the disconnect attention message.

Parameter (uses disconnectPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scDiscArrayPtr	long
	22	scArrayCount	word
	26	scCode	word
	28	scNumMinutes	word
	30	scFlags	word
	32	scMessagePtr	long
Fields	ioResult	Word result value: Result code.	
	scDiscArrayPtr	Longword input pointer: Points to the array of user name record IDs (UNRecID).	
	scArrayCount	Word input value: The number of elements in the array of user name record IDs (UNRecID).	
	scCode	Word input value: The server control code; always SCDisconnect (\$0004).	
	scNumMinutes	Word input value: The number of minutes until the users are disconnected, in the range 0-4094.	
	scFlags	Word input value: Shutdown flags, as follows:	
	bUNRFSendMsg	The message pointed to by scMessagePtr should accompany the disconnect.	

Note This feature is not supported by Macintosh File Sharing.

scMessagePtr	Longword input value: A pointer to a Str199 containing the message sent to the workstations.
--------------	--

Note This feature is not supported by Macintosh File Sharing.

Result Codes	noErr	0	No error.
	AlreadyShuttingDown	-1	The server is already shutting down.
	AlreadyDisconnecting	-2	The server is already disconnecting.
	paramErr	-50	The server is not running, scNumMinutes is out of range, an unknown bit is set in scFlags, or a UNRecID is invalid.

SCDisconnectVolUsers

SCDisconnectVolUsers disconnects any users who have any of the specified volumes mounted. In addition, this call prevents any new users from mounting the volumes. Calling SCCancelShutDown cancels the shutdown in progress and re-enables the mounting of volumes.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses disconnectPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scDiscArrayPtr	long
	22	scArrayCount	word
	26	scCode	word
	28	scNumMinutes	word
	30	scFlags	word
	32	scMessagePtr	long

Fields	ioResult	Word result value: Result code.
of	scDiscArrayPtr	Longword input pointer: Points to the array longs containing the volume reference numbers specifying the volumes affected.
	scArrayCount	Word input value: The number of elements in the array of volume reference numbers.
	scCode	Word input value: The server control code; always SCDisconnectVolUsers (\$0012).
	scNumMinutes	Word input value: The number of minutes until the users are disconnected.
	scFlags	Word input value: Shutdown flag, as follows:
	bUNRFSendMsg	The message pointed to by scMessagePtr should accompany the disconnect.

	scMessagePtr		Longword input value: A pointer to a Str199 containing the message sent to the workstations.
Result Codes	noErr	0	No error.
	AlreadyShuttingDown	-1	The server is already shutting down.
	AlreadyDisconnecting	-2	The server is already disconnecting.
	paramErr	-50	The server is not running, scArrayCount is greater than scMaxVolumes as returned by SCGetSetupInfo, a volume reference number is not valid, scNumMinutes is out of range, or an unknown bit is set in scFlags.

SCGetExpFldr

SCGetExpFldr returns information about shared folders and volumes.

Note Macintosh File Sharing does not return fnfErr when there is no shared volume or folder at a particular index position. Instead, it returns noErr and takes no other action. To determine if a particular location is in use, set scVRefNum to zero before calling SCGetExpFldr. If scVRefNum is still zero after SCGetExpFldr is called, then there is no shared volume or folder at that particular index position.

!! WARNING SCNamePtr must be NIL when scIndex is negative. Otherwise, Macintosh File Sharing writes garbage into memory. See the comments in the sample function code listed under "SCGetExpFldr" in Chapter 1. !!

Parameter (uses standardPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scNamePtr	long
	22	scVRefNum	word
	24	scLogins	word
	26	scCode	word
	28	scIndex	word
	30	scDirID	long

Fields ioResult Word result value: Result code.

where scNamePtr Longword input pointer: Points to a Str13 the shared folder's AFP short name will be returned, or must contain NIL. If scIndex is

negative, then an empty Pascal string (') is returned.

number	scVRefNum	Word result value: Returns the reference (vRefNum) of the shared folder.
people	scLogins	Word result value: Returns the number of who have mounted this folder. (For real volumes, this parameter returns the total number of people who have mounted either the whole volume or any of its shared folders.)

Note This value is not returned under Macintosh File Sharing.

	scCode	Word input value: The server control code; always SCGetExpFldr (\$0006).
	scIndex	Word input value: The index into the list of shared folders. Use positive values to get shared folders (what non-superusers see). Use negative values to get shared volumes (what superusers see). Use SCGetSetupInfo to find the usable range for scIndex. scIndex must be in the range -MaxVolumes to MaxExpFolder. An scIndex of 0 is undefined.
ID	scDirID	Longword result value: Returns the directory (dirID) of the shared folder.
Result Codes	noErr	0 No error.
	fnfErr	-43 There is no shared folder at that index position.
	paramErr	-50 The server is not running, or scIndex is either 0 or out of range.
	afpObjectNotFound	-5018 scIndex is either 0 or out of range under Macintosh File Sharing.

SCGetServerEventProc

SCGetServerEventProc returns the head of the server event handler queue.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses serverEventPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scSEQEntryPtr	long
	26	scCode	word

Fields	ioResult		Word result value: Result code.
	scSEQEntryPtr		Longword result pointer: Returns a pointer to an operating system queue header (QHdr) of the server event handler queue. The first server event handler in the handler queue, if any, is at QHdrPtr(scSEQEntryPtr)^.qhead.
	scCode		Word input value: The server control code; always SCGetServerEventProc (\$000D).
Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.

SCGetServerStatus

SCGetServerStatus returns server status information.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses statusPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scNamePtr	long
	26	scCode	word
	28	scServerFlags	word
	30	scNumSessions	word
	32	scUserListModDate	long
	36	scActivity	word
	38	scVollistModDate	long

!! WARNING Do not assume that scUserListModDate or scVollistModDate will be in any particular time base. The formats of these parameters are subject to change. Use them only as an indication that the user list or volume list has changed. !!

Fields	ioResult		Word result value: Result code.
	scNamePtr		Longword input pointer: Points to a Str31 where the current server name (the name workstations will see) will be returned, or must contain NIL.
	scCode		Word input value: The server control code; always SCGetServerStatus (\$000A).
	scServerFlags		Word result value: Server flag, as follows:

	bJBSEnabled		Set if Apple II boot service is enabled. All other bits are reserved.
	scNumSessions		Word result value: The number of currently opened sessions.
	scUserListModDate		Longword result value: The last date and time (DateTime) that the user list was modified. (This value is helpful in minimizing the amount of updating needed by a monitoring application that updates some user list.)
	scActivity		Word result value: The server activity, in percent (5%-100%).
	scVollistModDate		Longword result value: The last time (TickCount) that the volume list was modified. (This value is helpful in minimizing the amount of updating needed by a monitoring application that updates some volume list.)
Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.

SCGetSetupInfo

SCGetSetupInfo returns server setup information. The setup information record (SetupInfoRec) is defined in the server control calls interface file. (See Appendix B for a listing of the interface file. SetupInfoRec appears in the "Server Control Data Types" section.)

Parameter (uses setupPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scSetupPtr	long
	22	scMaxVolumes	word
	24	scMaxExpFolders	word
	26	scCode	word
	28	scCurMaxSessions	word

Fields	ioResult	Word result value: Result code.
	scSetupPtr	Longword input pointer: Points to the setup information record (SetupInfoRec) where the server setup information will be returned, or must contain NIL.
	scMaxVolumes	Word result value: Returns the maximum number of

volumes supported by the server.

Note This value is not returned under Macintosh File Sharing. (The maximum number of volumes supported under Macintosh File Sharing is 10.)

scMaxExpFolders Word result value: Returns the maximum number of shared folders supported by the server.

Note This value is not returned under Macintosh File Sharing. (The maximum number of folders supported under Macintosh File Sharing is 10.)

scCode Word input value: The server control code; always SCGetSetupInfo (\$0007).

scCurMaxSessions Word result value: Returns the maximum number of logins currently allowed.

Note This value is not returned under Macintosh File Sharing. Use SIMaxLogins (equal to 11).

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.

SCGetUserMountInfo

SCGetUserMountInfo returns information about how a user is using a particular volume. For a shared folder (that is, if the value of scVRefNum is positive), these values are for that shared folder only. For a real volume (that is, if the value of scVRefNum is negative), these values represent totals for all shared folders on the volume.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses volMountedPB variant of SCParamBlockRec)

Block	16	ioResult	word
	22	scVRefNum	word
	26	scCode	word
	28	scFilesOpen	word
	30	scWriteableFiles	word
	32	scUNRecID	long
	36	scMounted	byte
	37	scMountedAsOwner	byte

Fields ioResult Word result value: Result code.

scVRefNum Word input value: The volume specification or shared folder specification.

scCode	Word input value: The server control code; always SCGetUserMountInfo (\$0014).
scFilesOpen	Word result value: Returns the total number of files the user has open on the volume or shared folder.
scWriteableFiles	Word result value: Returns the total number of files the user has open for write access on the volume or shared folder.
scUNRecID	Longword input value: Specifies the user name record ID (UNRecID).
scMounted	Byte result value: Returns TRUE if the user has this volume mounted.
scMountedAsOwner	Byte result value: For real volumes only, returns TRUE if the user has the whole volume mounted by virtue of being a superuser.

Result Codes	noErr	0	No error.
	nsvErr	-35	No such volume with this reference number (scVRefNum).
	paramErr	-50	The server is not running, the user name record ID (scUNRecID) is invalid, or the volume reference number (scVRefNum) is out of range.

SCGetUserNameRec

SCGetUserNameRec retrieves statistics on a connected user, and can be used to enumerate all connected users.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses userInfoPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scNamePtr	long
	26	scCode	word
	28	scPosition	long
	32	scUNRecID	long
	36	scUserID	long
	40	scLoginTime	long
	44	scLastUseTime	long
	48	scSocketNum	long

Fields	ioResult	Word result value: Result code.
	scNamePtr	Longword result pointer: Points to a Str31 where the user name will be copied, or must contain NIL.
	scCode	Word input value: The server control code; always SCGetUserNameRec (\$0013).
	scPosition	Longword input/result value: Specifies the position in the list of users. Set scPosition to zero to retrieve the first user. Use the value returned in scPosition to retrieve the next user.
	scUNRecID	Longword result value: Returns the user name record ID (UNRecID).
	scUserID	Longword result value: Returns the user ID (UserID).
	scLoginTime	Longword result value: Returns the time at which the user logged in.
	scLastUseTime	Longword result value: Returns the time at which the user last accessed the server.
	scSocketNum	Longword result value: Returns the AppleTalk network address this user is connected from. The value is returned in an AddrBlock record.

Result Codes	noErr	0	No error.
	fnfErr	-43	There are no more users to enumerate.
	paramErr	-50	The server is not running, a UNRecID is invalid, or scPosition is out of range.

SERVER CONTROL CALL REFERENCE

SCInstallServerEventProc

SCInstallServerEventProc installs a server event object in the server event handler queue.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses serverEventPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scSEQEntryPtr	long
	26	scCode	word

Fields ioResult Word result value: Result code.

scSEQEntryPtr Longword input pointer: Points to the tSEQEntry server event object to be installed in the server event handler queue.

scCode Word input value: The server control code; always SCInstallServerEventProc (\$000B).

Result Codes

noErr	0	No error.
paramErr	-50	The server is not running.
afpMiscErr	-5014	There are already 15 server event handlers (the maximum) in the server event handler queue.

SCPollServer

SCPollServer provides information about the current status of the file server.

Parameter (uses pollServerPB variant of SCParamBlockRec)

Block	16	ioResult	word
	26	scCode	word
	28	scServerState	word
	30	scDisconnectState	word
	32	scServerError	word
	34	scSecondsLeft	long

Fields

ioResult	Word result value: Result code.
scCode	Word input value: The server control code; always SCPollServer (\$0005).
scServerState	Word result value: The state of the server, as follows:
\$0000	0-29 seconds before shutdown; Network Setup message says "Less than a minute."
\$0001	30-89 seconds before shutdown; Network Setup message says "About a minute."
\$0002-\$0FFE	(scServerState*60) - 30 to (scServerState*60) + 29 seconds before shutdown; Network Setup message says "About scServerState minutes."
SCPSRunning	Server running normally.
SCPSStartingUp	Server is in the process of starting up.

SCPSJustDisabled Server was just disabled and there was no startup error.

SCPSDisabledwErr Server is disabled and there is an "SE" error in scServerError.

SCPSSleeping Server is temporarily disabled.

Note This result is not returned by Macintosh File Sharing.

scDisconnectState Word result value: The state of the server disconnect, as follows:

\$0000 0-29 seconds before disconnect; Network Setup message says "Less than a minute."

\$0001 30-89 seconds before disconnect; Network Setup message says "About a minute."

\$0002-\$0FFE (scDisconnectState*60) - 30 to (scDisconnectState*60) + 29 seconds before disconnect; Network Setup message says "About scDisconnectState minutes."

SCPDNotDisconnecting Server not disconnecting some user or group of users.

scServerError Word result value: If scServerState = SCPSDisabledwErr then scServerError contains one of the following values:

SENoUGFileOpenErr The Users & Groups Data File could not be opened.

SENoRealVolsErr There are no volumes for the file server to use.

SEInsuffMFMemErr There was not enough memory available to start the file server.

SECantRegNameErr The file server's name could not be registered on the AppleTalk network.

SECantFindExtnFolder The file server could not be started because the Extensions folder could not be found.

SEUnExATalkErr An unexpected AppleTalk error occurred.

SENoMachineName A machine name is required.

SECantFindFSExtn The file server could not be started because the File Server Extension or File Sharing Extension could not be found.

SEATalkOffErr AppleTalk is turned off.

SEAppleTalkErr AppleTalk could not be activated.

SENoInitRunErr The File Server Extension or File Sharing Extension is not installed in the Server Folder.

SESysTooOldErr	The System file is too old for AppleShare File Server 3.0.
SEInsuffAppMemErr	There was not enough memory for the file server to startup.
SEBadConfigErr	The file server encountered a problem with the current configuration.
SENoDTOnStartupErr	The desktop database on the startup volume could not be opened.
SEDupNameErr	Duplicate-name error occurred when server was registering. Please choose another name.
21-29	Reserved by Apple.
scSecondsLeft	Longword result value: Returns the number of seconds left before the shutdown or disconnect. Zero is returned if no shutdown or disconnect in progress. This value is undefined if the server is disabled (not running).

Note This feature is not implemented under Macintosh File Sharing.

Result Codes	noErr	0	No error.
--------------	-------	---	-----------

SCRemoveServerEventProc

SCRemoveServerEventProc removes a server event object from the server event handler queue.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses serverEventPB variant of SCPParamBlockRec)

Block	16	ioResult	word
	18	scSEQEntryPtr	long
	26	scCode	word

Fields	ioResult	Word result value: Result code.
	scSEQEntryPtr	Longword input pointer: Points to the tSEQEntry server event object to be removed from the server event handler queue.
	scCode	Word input value: The server control code; always SCRemoveServerEventProc (\$000C).

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.
	afpMiscErr	-5014	There are no server event objects, or this server event object is not the server event handler queue.

SCSendMessage

SCSendMessage sends a server message to every user whose user name record ID (UNRecID) is contained in the array pointed to by scDiscArrayPtr.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses disconnectPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scDiscArrayPtr	long
	22	scArrayCount	word
	26	scCode	word
	30	scFlags	word
	32	scMessagePtr	long

Fields ioResult Word result value: Result code.

scDiscArrayPtr Longword input pointer: Points to the array of user name record IDs (UNRecID).

scArrayCount Word input value: The number of elements in the array of user name record IDs (UNRecID).

scCode Word input value: The server control code; always SCSendMessage (\$0009).

scFlags Word input value: The following bit must be set:

bUNRFSendMsg There is a message pointed to by scMessagePtr.

scMessagePtr Longword input value: A pointer to a Str199 containing the message sent to the workstations.

Result Codes	noErr	0	No error.
	AlreadyShuttingDown	-1	The server is already shutting down.
	AlreadyDisconnecting	-2	The server is already disconnecting.
	paramErr	-50	The server is not running or a UNRecID is invalid.

SCServerVersion

SCServerVersion returns the server version information.

!! WARNING Macintosh File Sharing does not return a valid value for scServerVersion if the server is not running. !!

Parameter (uses versionPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scExtNamePtr	long
	26	scCode	word
	28	scServerType	word
	30	scServerVersion	word

Fields	ioResult	Word result value: Result code.
	scExtNamePtr	Longword result pointer: Points to a Str31 where the server application name (the name of the activeINIT) will be returned, or must contain NIL.
	scCode	Word input value: The server control code; always SCServerVersion (\$000E).
	scServerType	Word result value: Returns the server type, as follows:
	\$0000	Macintosh File Sharing
	\$0001	AppleShare File Server
	scServerVersion	Word result value: Returns the server version, as follows:
	\$0030	File Sharing Extension, version 7.0.2

Result Codes	noErr	0	No error.
--------------	-------	---	-----------

SCSetCopyProtect

SCSetCopyProtect is called by the AppleShare Admin application or some other program executing locally on the server computer when the program wants to set the copy-protect status of a file.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses standardPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scNamePtr	long
	22	scVRefNum	word
	26	scCode	word
	30	scDirID	long

Fields ioResult Word result value: Result code.

scNamePtr	Longword input pointer: Points to the filename.
scVRefNum	Word input value: The volume specification.
scCode	Word input value: The server control code; always SCSetCopyProtect (\$0010).
scDirID	Longword input value: The parent directory ID.

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running.

SCSetCopyProtect may also return errors returned by the PBGetCatInfo and PBSetCatInfo routines.

SCSetSetupInfo

SCSetSetupInfo sets the server setup information. All changes take effect immediately except those affecting the Volume Info window and the Connected Users window. Specifically, changes to the following four fields of the setup information record (SetupInfoRec) do not take effect until the next time the AppleShare File Server application starts up:

- SIVolInfoLocation: Point; { location of Volume Info window }
- SIVolInfoVisible: Boolean; { is Volume Info window visible? }
- SIUserInfoLocation: Point; { location of Connected Users window }
- SIUserInfoVisible: Boolean; { is Connected Users window visible? }

The setup information record (SetupInfoRec) is defined in the server control calls interface file. (Appendix B contains a listing of the interface file. SetupInfoRec appears in the "Server Control Data Types" section.)

Parameter (uses setupPB variant of SCParamBlockRec)

Block	16	ioResult	word
	18	scSetupPtr	long
	26	scCode	word

Fields ioResult Word result value: Result code.

scSetupPtr	Longword input pointer: Points to a valid pre-allocated server setup information record (SetupInfoRec).
scCode	Word input value: The server control code; always SCSetSetupInfo (\$0008).

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not running,

scSetupPtr is NIL, or SetupInfoRec contains a value that is out of range.

SCShutDown

SCShutDown shuts down the file server and sends a shutdown attention message to all connected users.

Note Macintosh File Sharing does not support the shutdown attention message.

!! IMPORTANT The AppleShare File Server application automatically quits if the AppleShare File Server 3.0 is shut down by the SCShutDown call. !!

Parameter (uses disconnectPB variant of SCPParamBlockRec)

Block	16	ioResult	word
	26	scCode	word
	28	scNumMinutes	word
	30	scFlags	word
	32	scMessagePtr	long

Fields ioResult Word result value: Result code.

scCode	Word: input value: The server control code, always be SCShutDown (\$0002).
scNumMinutes	Word input value: The number of minutes 4094.
scFlags	Word input value: Shutdown flag, as follows:
bUNRFSendMsg	The message pointed to by scMessagePtr should accompany the disconnect.

Note This feature is not supported by Macintosh File Sharing.

scMessagePtr	Longword input value: A pointer to a Str199 containing the message sent to the workstations.
--------------	--

Note This feature is not supported by Macintosh File Sharing.

Result Codes	noErr	0	No error.
	AlreadyShuttingDown	-1	The server is already shutting down.
	AlreadyDisconnecting	-2	The server is already disconnecting.
	paramErr	-50	The server is not running, scNumMinutes is out of

range, or an unknown bit is set in scFlags.

SCSleepServer

SCSleepServer shuts down the file server temporarily. This call has the same parameters as SCShutdown except that once the server has shut down, the AppleShare File Server application does not quit, and the server can be restarted by means of the SCWakeServer call (assuming that no SCShutdown call is made while the server is asleep). SCSleepServer fails if the server is starting up.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses disconnectPB variant of SCParamBlockRec)

Block	16	ioResult	word
	26	scCode	word
	28	scNumMinutes	word
	30	scFlags	word
	32	scMessagePtr	long

Fields ioResult Word result value: Result code.

code;	scCode	Word input value: The server control always SCSleepServer (\$0016).
	scNumMinutes	Word input value: The number of minutes until server shutdown, in the range 0-4094.
	scFlags	Word input value: Shutdown flag, as follows:
	bUNRFSendMsg	The message pointed to by scMessagePtr should accompany the disconnect.
	scMessagePtr	Longword input value: A pointer to a Str199 containing the message sent to the workstations.

Result Codes	noErr	0	No error.
	AlreadyShuttingDown	-1	The server is already shutting down.
	AlreadyDisconnecting	-2	The server is already disconnecting.
	paramErr	-50	The server is not running, scNumMinutes is out of range, or an unknown bit is set in scFlags.

SCStartServer

SCStartServer starts the file server.

!! IMPORTANT The AppleShare File Server 3.0 is normally started by the AppleShare File Server application. When the AppleShare File Server application is launched, it checks to see if the file server is running. If it is, the AppleShare File Server application assumes its role as the file server's user interface. If the file server is not running, the AppleShare File Server application starts the server by calling SCStartServer before assuming its role as user interface.

If a server addition starts the AppleShare File Server 3.0 by calling SCStartServer, the file service starts up, but the AppleShare File Server application (the user interface) does not. Unless it provides the functionality of the AppleShare File Server application, your program should probably not call SCStartServer to start the AppleShare File Server 3.0. Instead, start the file server in the usual way -- by launching the AppleShare File Server application. !!

Parameter (uses startPB variant of SCParamBlockRec)

Block	16	ioResult	word
	26	scCode	word
	28	scStartSelect	word
	30	scEventSelect	word

Fields ioResult Word result value: Result code.

scCode	Word input value: The server control code, which must always be SCStartServer (\$0000).
scStartSelect	Word input value: Determines the server to start, as follows:
kCurInstalled	Use this value to start up the currently installed server, either an AppleShare File Server 3.0 or Macintosh File Sharing.
scEventSelect	Word input value: Always kFinderExtn

Result Codes	noErr	0	No error.
	paramErr	-50	The server is already running.

Other errors from the launching of the server -- such as fnfErr and memFullErr -- may also be returned.

SCWakeServer

SCWakeServer reactivates a server that has been "put to sleep" (temporarily shut down with the SCSleepServer call). SCWakeServer will only start a server that has been put to sleep; it will not start a server that has been shut down by the SCShutdown call. No parameters are needed, since this call uses the parameters passed in by the original StartServer trap.

Note This call is not supported by Macintosh File Sharing.

Parameter (uses standardPB variant of SCParamBlockRec)

Block	16	ioResult	word
	26	scCode	word

Fields ioResult Word result value: Result code.

scCode Word input value: The server control code; always SCWakeServer (\$0015).

Result Codes	noErr	0	No error.
	paramErr	-50	The server is not sleeping.

Other errors from the launching of the server -- such as fnfErr and memFullErr -- may also be returned.

APPENDIX A

MACINTOSH FILE SHARING SERVER CONTROL CALLS

Macintosh File Sharing supports a subset of the AppleShare File Server 3.0 server control calls. This appendix lists the calls available with Macintosh File Sharing and discusses the differences between using server control calls with the AppleShare File Server 3.0 and using them with Macintosh File Sharing.

Macintosh File Sharing supports the following server control calls:

- SCCancelShutDown
- SCDisconnect
- SCGetExpFldr
- SCGetSetupInfo
- SCPollServer
- SCServerVersion
- SCSetSetupInfo
- SCShutDown
- SCStartServer

Of the server control calls that are supported, some of these calls behave differently under Macintosh File Sharing than they do under the AppleShare File Server 3.0. The sections that follow explain those differences.

SCDisconnect

The SCDisconnect call does not send disconnect attention messages under Macintosh File Sharing.

SCGetExpFldr

With Macintosh File Sharing, your program should call SCGetExpFldr as shown in the sample function in the section "SCGetExpFldr" in Chapter 1.

SCNamePtr must be NIL when scIndex is negative. Otherwise, Macintosh File Sharing writes garbage into memory. See the comments in the sample function code listed in the section "SCGetExpFldr" in Chapter 1.

Macintosh File Sharing does not return fnfErr when there is no shared volume or folder at a particular index position. Instead, it returns noErr and takes no other action. To determine if a particular location is in use, set scVRefNum to zero before calling SCGetExpFldr. If scVRefNum is still zero after SCGetExpFldr is called, then there is no shared volume or folder at that particular index position.

The SCGetExpFldr call does not return scLogins under Macintosh File Sharing.

SCGetSetupInfo

The SCGetSetupInfo call does not return the following results under Macintosh File Sharing:

- scMaxVolumes (Use the value 10.)
- scMaxExpFolders (Use the value 10.)
- scCurMaxSession (Use SIMaxLogins, which is equal to 11.)

The SCGetSetupInfo call also does not use the following fields of the setup information record (SetupInfoRec):

- SIVolInfoVisible
- SIUserInfoLocation
- SIUserInfoVisible
- IShutdownMins
- SISpare
- SILoginMsg

SCPollServer

The SCPollServer call does not return the following values under Macintosh File Sharing:

- the SCPSSleeping value of the scServerState result
- scSecondsLeft

SCServerVersion

Macintosh File Sharing does not return a valid value for SCServerVersion if the server is not running.

SCSetSetupInfo

The SCSetSetupInfo call does not use the following fields of the setup information record (SetupInfoRec):

- SIVolInfoVisible
- SIUserInfoLocation
- SIUserInfoVisible
- SIShutdownMins
- SISpare
- SILoginMsg

SCShutdown

The SCShutdown call does not send shutdown attention messages under Macintosh File Sharing.

APPENDIX B

INTERFACE FILES

This appendix lists the server control and server event interface files.

Server control interface file

The ServerControlINTF file contains all of the definitions for the server control calls used to control Macintosh File Sharing and the AppleShare File Server 3.0.

Server control constants

CONST

ServerDispatch = \$A094; {server control dispatch trap}

{scCode values}

SCStartServer = 0;

SCShutdown = 2;

SCCancelShutdown = 3;

SCDisconnect = 4;

SCPollServer = 5;

```

SCGetExpFldr          = 6;
SCGetSetupInfo       = 7;
SCSetSetupInfo       = 8;
SCSendMessage        = 9;
SCGetServerStatus    = 10;
SCInstallServerEventProc = 11;
SCRemoveServerEventProc = 12;
SCGetServerEventProc = 13;
SCServerVersion      = 14;
SCSetCopyProtect     = 16;
SCClrCopyProtect     = 17;
SCDisconnectVolUsers = 18;
SCGetUserNameRec     = 19;
SCGetUserMountInfo  = 20;
SCWakeServer         = 21;
SCSleepServer        = 22;

{scFlags bits and masks for disconnectPB}
bUNRFSendMsg         = 13;    {send a message}
UNRFSendMsgMask      = $2000; {send a message}
{error codes returned from an SCDisconnect trap}
AlreadyShuttingDown = -1; {the server is already }
{ shutting down}
AlreadyDisconnecting = -2; {the server is already }
{ disconnecting}
{the server types returned in scServerType}
MFSType = $0000; {Macintosh File Sharing}
AFSType = $0001; {AppleShare File Server}
{the server versions returned in scServerVersion}
{ $0030 = File Sharing Extension, versions 7.0}

```

```
{          and 7.0.1}
{ $0031 = File Server Extension, version 3.0}
{ $0032 = File Sharing Extension, version 7.0.2}
{some random constants for SCStartServer}
kCurInstalled = 0; {use currently installed server}
kFinderExtn   = 0; {use the Finder extension}
{scServerFlags bits returned by SCGetServerStatus}
bJBSEnabled = 12; {Apple II boot service is enabled}
{"SCPS" integers returned by SCPollServer in }
{ scServerState}
{0 means: 0 <= seconds left < 30}
{1 means: 30 <= seconds left < 90}
{2 thru 4094 mean: x*60-30 <= seconds left < x*60+30}
SCPSRunning      = -1; {Server running normally}
SCPSStartingUp   = -2; {Server starting up}
SCPSJustDisabled = -3; {Server disabled and there was }
{ no startup error}
SCPSDisabledwErr = -4; {Server disabled and there is }
{ a Server Error (SE) error in }
{ scServerError}
SCPSSleeping     = -5; {Server temporarily disabled}
{"SCPD" integers returned by SCPollServer in }
{ scDisconnectState}
{0 means: 0 <= seconds left < 30}
{1 means: 30 <= seconds left < 90}
{2 thru 4094: x*60-30 <= seconds left < x*60+30}
SCPDPNotDisconnecting = -1; {Server is not }
{ disconnecting any users}
{Server Errors ("SE") error codes returned by }
```

```
{ SCPollServer in scServerError}
SENoUGFileOpenErr    = 1;  {The Users & Groups Data }
{ File could not be opened}
SENoRealVolsErr      = 2;  {There are no volumes for }
{ the file server to use}
SEInsuffMFMemErr     = 4;  {There was not enough }
{ memory available to start }
{ the file server}
SECantRegNameErr     = 5;  {The file server's name }
{ could not be registered }
{ on the AppleTalk Network}
SECantFindExtnFolder = 6;  {The file server could not }
{ be started because the }
{ Extensions folder could }
{ not be found}
SEUnExATalkErr       = 7;  {An unexpected AppleTalk }
{ error occurred}
SENoMachineName      = 8;  {You must have a machine }
{ name}
SECantFindFSExtn     = 9;  {The file server could not }
{ be started because the }
{ File Server Extension or }
{ File Sharing Extension }
{ could not be found}
SEATalkOffErr        = 10; {Appletalk is turned off}
SEAppleTalkErr       = 11; {AppleTalk could not be }
{ activated}
SENoInitRunErr       = 12; {The file server could not }
{ be started because the }
```

```

{ File Server Extension or }
{ File Sharing Extension }
{ could not be found}
SESysTooOldErr      = 13; {The System File is too old }
{ for AppleShare (v3.0)}
SEInsuffAppMemErr   = 14; {There was not enough }
{ memory for the file server }
{ to startup}
SEBadConfigErr      = 15; {The file server encountered }
{ a problem with the current }
{ configuration}
SENoDTOnStartupErr = 16; {The Desktop database on }
{ volume the startup volume }
{ could not be opened}
SEDupNameErr        = 17; {Duplicate name error when }
{ registering}
Server control data types
TYPE
{string used for server messages}
tLoginMsg = STRING[199];
{The SetupInfoRec used by the SCGetSetupInfo }
{ and SCSetSetupInfo calls.}
SetupInfoRecPtr = ^SetupInfoRec;
SetupInfoRec =
RECORD
SIVersion:      Integer;    {SetupInfoRec version}
{ 1 for File Sharing }
{ and AppleShare 3.0}
SIFlags:        Integer;    {0 for File Sharing }

```

```

{ and AppleShare 3.0}
SIMaxLogins:      Integer;      {1..11 for File }
{ Sharing; 1..121 }
{ for AppleShare 3.0}
SISrvrUsageLimit: Integer;      {10 to 100 (percent)}
{All remaining fields in record are only used by the }
{ AppleShare 3.0 file server}
SIVolInfoLocation: Point;      {location of Volume }
{ Info window}
SIVolInfoVisible: Boolean;      {is Volume Info }
{ window visible?}
SIUserInfoLocation: Point;      {location of }
{ Connected Users }
{ window}
SIUserInfoVisible: Boolean;      {is Connected Users }
{ window visible?}
SIShutDownMins:   Integer;      {default minutes }
{ until shutdown}
SISpare:          ARRAY[1..17] OF Integer;
{reserved}
SILoginMsg:       tLoginMsg;    {the current Login }
{ message}
END;

startParam =
RECORD
reserved:        LongInt;
reserved2:       Integer;
reserved3:       Integer;
scCode:         Integer;

```

```
scStartSelect: Integer;
scEventSelect: Integer;
reserved4:     ARRAY[1..4] OF LongInt;
END;

disconnectParam =
RECORD
scDiscArrayPtr: LongIntPtr;
scArrayCount:   Integer;
reserved:       Integer;
scCode:         Integer;
scNumMinutes:   Integer;
scFlags:        Integer;
scMessagePtr:   StringPtr;
END;

pollServerParam =
RECORD
reserved:       LongInt;
reserved2:      Integer;
reserved3:      Integer;
scCode:         Integer;
scServerState:  Integer;
scDisconnectState: Integer;
scServerError:  Integer;
scSecondsLeft:  LongInt;
END;

standardParam =
RECORD
scNamePtr: StringPtr;
scVRefNum: Integer;
```

```
scLogins: Integer;
scCode: Integer;
scIndex: Integer;
scDirID: LongInt;
END;

setupParam =
RECORD
scSetupPtr: SetupInfoRecPtr;
scMaxVolumes: Integer;
scMaxExpFolders: Integer;
scCode: Integer;
scCurMaxSessions: Integer;
END;

statusParam =
RECORD
scNamePtr: StringPtr;
reserved2: Integer;
reserved3: Integer;
scCode: Integer;
scServerFlags: Integer;
scNumSessions: Integer;
scUserListModDate: LongInt;
scActivity: Integer;
scVolListModDate: LongInt;
END;

serverEventParam =
RECORD
scSEQEntryPtr: Ptr;
reserved2: Integer;
```

```
reserved3:    Integer;
scCode:       Integer;
END;
versionParam =
RECORD
scExtNamePtr: StringPtr;
reserved2:    Integer;
reserved3:    Integer;
scCode:       Integer;
scServerType: Integer;
scServerVersion: Integer;
END;
userInfoParam =
RECORD
scNamePtr:    StringPtr;
reserved2:    Integer;
reserved3:    Integer;
scCode:       Integer;
scPosition:   LongInt;
scUNRecID:    LongInt;
scUserID:     LongInt;
scLoginTime:  LongInt;
scLastUseTime: LongInt;
scSocketNum:  AddrBlock;
END;
volMountedParam =
RECORD
reserved:     Ptr;
scVRefNum:    Integer;
```

```
reserved3:      Integer;
scCode:         Integer;
scFilesOpen:    Integer;
scWritableFiles: Integer;
scUNRecID:      LongInt;
scMounted:      Boolean;
scMountedAsOwner: Boolean;
END;

SCParamBlockPtr = ^SCParamBlockRec;
SCParamBlockRec =
RECORD
qLink:          QElemPtr;
qType:          INTEGER;
ioTrap:         INTEGER;
ioCmdAddr:     Ptr;
ioCompletion:   ProcPtr;
ioResult:      OSErr;
CASE Integer OF
1: (startPB:      startParam);
2: (disconnectPB: disconnectParam);
3: (pollServerPB: pollServerParam);
4: (standardPB:   standardParam);
5: (setupPB:      setupParam);
6: (statusPB:     statusParam);
7: (serverEventPB: serverEventParam);
8: (versionPB:    versionParam);
9: (userInfoPB:   userInfoParam);
10: (volMountedPB: volMountedParam);
END;
```

Server control routine

```
FUNCTION SyncServerDispatch (pb: SCPParamBlockPtr): OSErr;
```

Server event interface file

The ServerEventINTF file contains all of the definitions for the server event mechanism.

Server event constants

CONST

{Constants used in the tSEQEntry}

{The SEeventFlag bits in tSEQEntry specify when the }
{ server event handler would like to be called}

{SEeventFlag bits}

```
bCSEHAFPIInDoRequest    = 0;
```

{An AFP call is starting up }

{ (in DoRequest, about to be dispatched)}

```
bCSEHAFPIInSendResponse = 1;
```

{An AFP call has completed}

{ (in SendResponse, about to send out the response)}

```
bCSEHServerBusy        = 2;
```

{A new session is being denied because the server }

{ is busy (socket starvation event)}

```
bCSEHServerShutdown    = 3;
```

{The server just shut down}

```
bCSEHServerControlCall = 4;
```

{A server control call has just been completed}

{ NOTE: The following server control calls do not }

{ cause a bCSEHServerControlCall server event: }

{ SCStartServer, SCInstallServerEventProc, }

{ SCRemoveServerEventProc, SCGetServerEventProc, }

{ SCServerVersion, and SCWakeServer}

{ You can use the bCSEHServerStartup server event }

```
{ to detect server starts and wakeups}
bCSEHShare          = 5;
{An HFS Share trap has just been completed}
bCSEHUnShare        = 6;
{An HFS UnShare trap has just been completed}
bCSEHSetDirAccess   = 7;
{An HFS SetDirAccess trap has just been completed}
bCSEHServerNameChange = 8;
{An attempt was made to change the server name }
{ (the attempt may or may not have been successful)}
bCSEHVolumePrep     = 9;
{A new volume was just prepared for use with }
{ AppleShare}
bCSEHVolumeUnmount  = 10;
{A volume unmount was attempted on an AppleShare }
{ volume }
bCSEHServerStartup  = 11;
{The server just successfully started up}
bCSEHSessionTornDown = 12;
{A user's session was torn down because any one of }
{ a number of reasons, including a disconnect, }
{ server shutdown, timeout, or workstation initiated }
{ close session}
bCSEHOutOfSequence  = 13;
{A packet was received out of sequence; the session }
{ may be a zombie}
bCSEHWksClosedSession = 14;
{A workstation closed its ASP session }
{ (i.e., it logged out)}
```

```

bCSEHSessionTimedOut    = 15;
{A workstation's session timed out}

bCSEHSrvrClosedSession = 16;
{The server has closed a workstation's session}
{When SEventFlag bits bCSEHAFPIInDoRequest or }
{ bCSEHAFPIInSendResponse are set, the bits in }
{ SEwhichAFPFlag determine which AFP calls will cause }
{ the server event handler to be called. }
{SEwhichAFPFlag bits}
{ bit 0 of SEwhichAFPFlag[1] - AFPCommand = 192 }
{                                     (afpAddIcon) }
{ bit 1 of SEwhichAFPFlag[1] - AFPCommand = 1  }
{ bit 2 of SEwhichAFPFlag[1] - AFPCommand = 2  }
{ ...                                     }
{ bit 31 of SEwhichAFPFlag[1] - AFPCommand = 31 }
{ bit 0 of SEwhichAFPFlag[0] - AFPCommand = 32 }
{ ...                                     }
{ bit 63 of SEwhichAFPFlag[0] - AFPCommand = 63 }
{When SEventFlag bit bCSEHServerControlCall is set, }
{ the bits in SEwhichSCFlag determine which server }
{ control calls will cause the server event handler to }
{ be called.}
{SEwhichSCFlag bits}
{ bit 0 of SEwhichSCFlag - scCode = 0  }
{ ...                                     }
{ bit 31 of SEwhichSCFlag - scCode = 31 }
{The maximum size of theBuffer in the ServerEventRecord}
BufferMax = 48;

Server event data types

```

TYPE

ServerEventRecordPtr = ^ServerEventRecord;

ServerEventRecord =

RECORD

theEventNumber: LongInt;

{the server event that's occurring; see the }

{ SEventFlag definitions above}

theServerTime: LongInt;

{ the server time (in Macintosh DateTime form)}

theResult: Integer;

{the result of the operation }

{ if theEventNumber = bCSEHAFPIInSendResponse: }

{ the AFP Error code to be returned }

{ if theEventNumber = bCSEHServerControlCall: }

{ the result of the server control call }

{ if theEventNumber = bCSEHShare, }

{ bCSEHUnShare, or bCSEHSetDirAccess: }

{ the result of the HFS call }

{ if theEventNumber = bCSEHServerNameChange: }

{ the result of a PRegisterName call, }

{ SECantRegNameErr, or SEDupNameErr }

{ if theEventNumber = bCSEHVolumeUnmount: }

{ noErr or fBsyErr (if volume is being used }

{ by a remote user) }

{ all other values of theEventNumber return noErr}

theBufSize: Integer;

{the number of bytes used in theBuffer}

theBuffer: PACKED ARRAY[1..BufferMax] OF Byte;

{ if theEventNumber = bCSEHAFPIInDoRequest or }

```

{  bCSEHAFPIInSendResponse: the first }
{  BufferMax bytes of the AFP packet }
{ if theEventNumber = bCSEHServerControlCall: }
{  the first BufferMax bytes of the }
{  SCPParamBlockRec }
{ if theEventNumber = bCSEHShare, }
{  bCSEHUnShare, bCSEHSetDirAccess, }
{  bCSEHVolumePrep, or bCSEHVolumeUnmount: }
{  the first BufferMax bytes of the }
{  HParamBlockRec }
{ if theEventNumber = bCSEHServerNameChange: }
{  the new server name (in a Pascal string) }
{ all other values of theEventNumber return a }
{  zero length buffer }
theNameStr:      Str31;
{the name of the file, if any; not always }
{ defined }
theAFPCommand:  Integer;
{ if theEventNumber = bCSEHAFPIInDoRequest or }
{  bCSEHAFPIInSendResponse, the AFP call}
{Note: If theEventNumber is bCSEHAFPIInDoRequest, }
{ bCSEHAFPIInSendResponse, bCSEHSessionTornDown, }
{ bCSEHOutOfSequence, bCSEHWksClosedSession, }
{ bCSEHSessionTimedOut, or bCSEHSrvrClosedSession,}
{ then theUNRecID theUNSUserID, theUserName, and }
{ theSocketAddress of the user that made the call }
{ are returned.}
theUNRecID:      LongInt;
{the UNRecID of the user that made the call}

```

```

theUNSUserID:      LongInt;
{the UserID of the user that made the call}
theUserName:       Str31;
{the name of the user that made the call}
{Note: If theEventNumber is bCSEHAFPIInDoRequest }
{ or bCSEHAFPIInSendResponse, then theVRefNum and }
{ theDirID will be returned if applicable to the }
{ AFP call}
theVRefNum:        Integer;
{the VRefNum of the volume upon which this }
{ operation was performed (not always applicable)}
theDirID:          LongInt;
{the DirID of the directory upon/within which this}
{ operation was performed (not always applicable)}
theSocketAddress: AddrBlock;
{the network address of the user's workstation in }
{ AddrBlock format}
END;
tSEQEntryPtr = ^tSEQEntry;
tSEQEntry =
RECORD
SEQentry:         ATQEntry;
{a regular AppleTalk Transition Queue entry}
SEeventFlag:      LongInt;
{specifies when the Server Event Handler would }
{ like to be called}
SEwhichAFPFlag:   ARRAY[0..1] OF LongInt;
{specifies which AFP calls will cause the Server }
{ Event Handler to be called}

```

```
SEwhichSCFlag: LongInt;  
{specifies which Server Control calls will cause }  
{ the Server Event Handler to be called}  
END;  
Application-defined routine  
PROCEDURE MyServerEventHandler (theSEQEntryPtr: tSEQEntryPtr;  
theSERecPtr: ServerEventRecordPtr);
```