AppleShare IP 6.2
Developer's Kit

Server Control Calls and Server Event Handling

Technical Publications

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

Chapter 1 Server Control Calls 11

About This Manual

Server control calls and server event handling are two features of the AppleShare IP 6.2 file server that allow Apple and third-party developers to modify and extend the capabilities of AppleShare file services. This manual is written for AppleShare developers and describes both server control calls and server event handling. Useful segments of sample code are included to help developers understand how to use the various calls. This manual also includes a reference section that provides the parameter block, field descriptions, and result codes for each server control call. Appendixes explain the differences between the server control calls available with Macintosh File Sharing and those available with the AppleShare IP 6.2 file server, and list the server control and server event interface files.

Conventions Used in This Manual

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

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

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

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

For More Information

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

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

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

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

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

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

For information on user authentication modules (UAMs), see the following publication:

- *AppleShare IP 6.2 Developer's Kit*: User Authentication Modules. Apple Computer, Inc.

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

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

CHAPTER 1

Server Control Calls

Contents

This chapter introduces the server control calls available with the AppleShare IP 6.2 file server and describes how server control calls interact with the main elements of file server software. The chapter presents several sample code segments and concludes with reference information for each server control call.

About Server Control Calls

Server control calls enable applications to monitor and control the major functions of the AppleShare IP 6.2 file server. 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, "Server Event Handling,") make it possible to create any number of services and utilities for the AppleShare IP 6.2 file server. 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 IP 6.2 file 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 manual refers to such programs and applications as **server additions**.

Note
Macintosh File Sharing supports a subset of the AppleShare IP 6.2 file server control calls. See Appendix A for a list of these calls. ◆

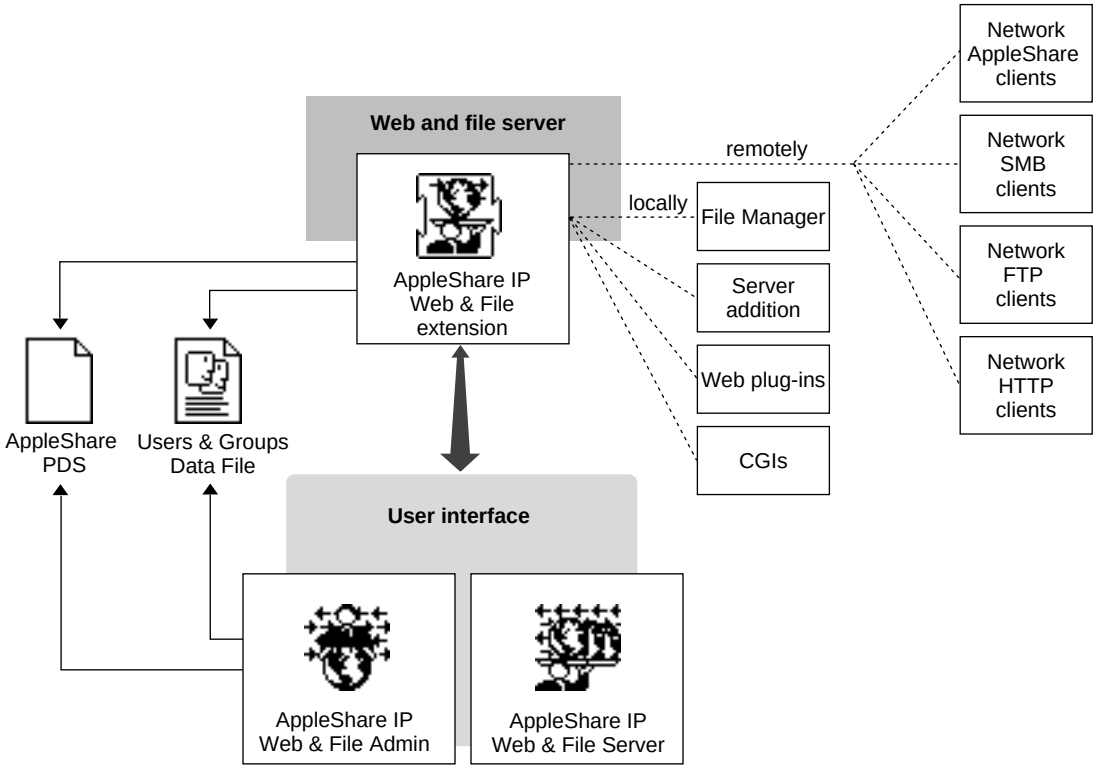
Main Elements of File Servers and Server Control Calls

This section describes the software components and data files that make up the AppleShare IP 6.2 file server and Macintosh File Sharing. Because the AppleShare IP 6.2 file server and Macintosh File Sharing perform similar functions, the components for each are similar and both use the same types of data files.

AppleShare IP 6.2 File Server Software Components

The AppleShare IP 6.2 file server is composed of a number of files, as shown in Figure 1-1. The AppleShare IP Web & File extension provides the actual functionality of the file server. The AppleShare IP Web & File Server and the AppleShare IP Web & File Admin applications provide the user interface for the server.

Figure 1-1 AppleShare IP 6.2 File Server Components



This section describes each of AppleShare IP 6.2 file server software components. The section “Data Files,” later in this chapter, describes the Users & Groups Data File and the AppleShare PDS file.

AppleShare IP Web & File Extension

The AppleShare IP Web & File extension contains the actual file server code. It is an extension of the system and resides in the Extensions folder. The AppleShare IP Web & File 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 AppleShare IP Web & File extension is launched, it runs as a background application.

The AppleShare IP Web & File extension contains no user interface of its own. The user interface is provided by the AppleShare IP Web & File Server and AppleShare IP Web & File Admin applications (described next). These applications communicate with the AppleShare IP Web & File 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 AppleShare IP Web & File extension communicates with remote clients through AppleTalk Filing Protocol (AFP), Server Message Block (SMB), File Transfer Protocol (FTP), and Hypertext Transfer Protocol (HTTP) sessions, and, locally, with shared volumes and files by means of Macintosh File Manager routines.

When the AppleShare IP Web & File extension is launched, it checks its environment, the Users & Groups Data File, and the desktop databases and AppleShare PDS files of appropriate volumes. (The AppleShare IP Web & File 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. If two volumes have the same name, the AppleShare IP Web & File extension only shares the first volume that it finds.

Once started, the AppleShare IP Web & File 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 volumes. The file server imposes access privilege constraints on AFP requests and implements some calls that are not implemented in the file system—such as those that govern byte-range locking, access privileges, and extended file access permissions.

AppleShare IP Manager

The AppleShare IP Manager provides a convenient way to start AppleShare IP 6.2 servers and to start server administration applications.

AppleShare IP Web & File Admin

Administrators can start the file server from the AppleShare IP Manager or by opening the AppleShare IP Web & File Admin application and choosing Start Web & File Server from Server menu. The AppleShare IP Web & File Admin application provides the interface for controlling and monitoring the file server while it is running, as well as the interface for defining users and groups for the server. The AppleShare IP Web & File Admin application also lets the administrator set preferences, set access privileges, and perform other administrative tasks for the file server. (See the *AppleShare IP Administrator's Manual* for more information about the administrative features of the AppleShare IP Web & File Admin application.)

The AppleShare IP Web & File Admin application communicates with the AppleShare IP Web & File extension by means of server control calls and file system calls. It communicates with the AppleShare Registry to store and retrieve information about server volumes and the users and groups defined for the server in the AppleShare PDS file and the Users & Groups Data File, respectively.

AppleShare IP Web & File Server

The AppleShare IP Web & File Server application provides another way to start and monitor the file server. The Server menu lets administrators unmount volumes, disconnect users, send messages to users, reset the cache, and set the greeting message. (See the *AppleShare IP Administrator's Manual* for more information about the features of the AppleShare IP 6.2 file server user interface.)

The AppleShare Web & File Server application communicates with the AppleShare IP Web & File extension primarily by means of server control calls.

AFP Clients

Computers with AppleShare client software installed can connect to the AppleShare IP Web & File extension. AppleShare clients communicate with the server through AFP sessions.

SMB Clients

Computers with Client for MS Networks software installed can connect to the AppleShare IP Web & File extension. These computers communicate with the server through SMB sessions.

FTP Clients

Computers with Transmission Control Protocol (TCP) software installed can use FTP to communication with the AppleShare IP Web & File extension.

HTTP Clients

Computers with Transmission Control Protocol (TCP) software installed can use a web browser to connect to the AppleShare IP Web & File extension through HTTP sessions.

File Manager

The Macintosh File Manager normally handles local requests for file access. While the file server is running, however, the AppleShare IP Web & File extension intercepts all file access calls from the File Manager.

Server Additions

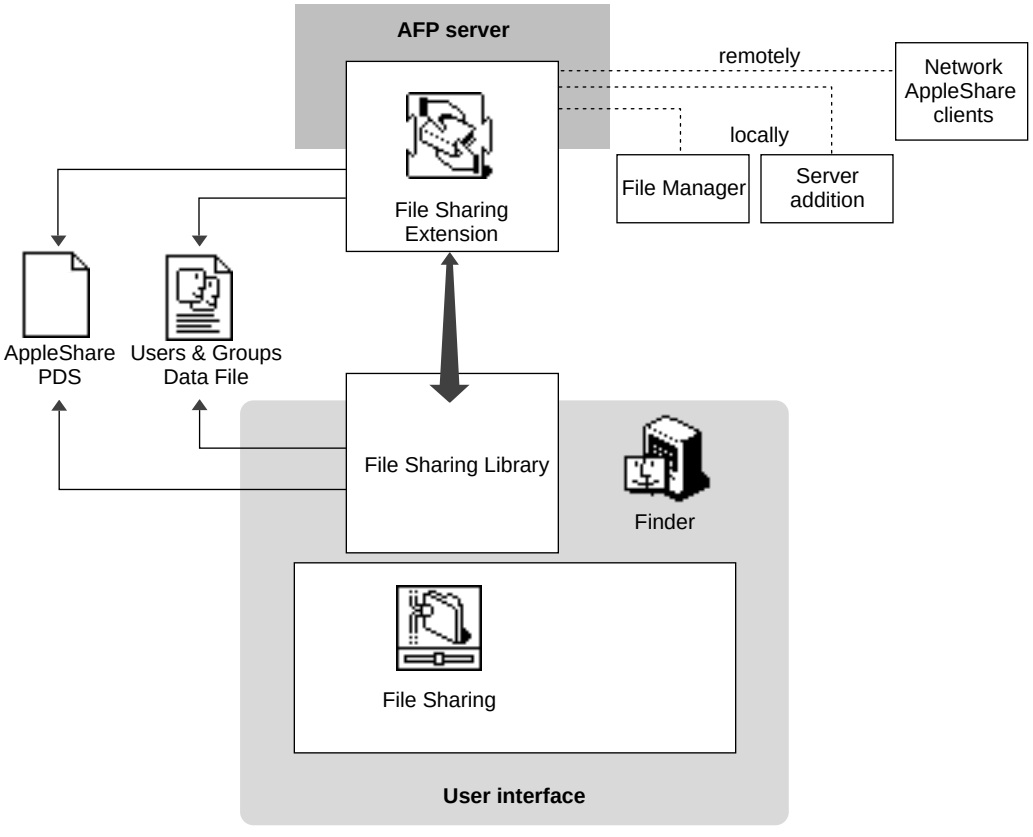
Applications, INITs, extensions, and other types of programs can access the AppleShare IP Web & File extension by using server control calls. A program that uses server control calls is referred to as a server addition. This manual tells you how to create server additions by using server control calls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

Macintosh File Sharing Software Components

Like the AppleShare IP 6.2 file server, Macintosh File Sharing is composed of a number of parts distributed across several files in the System Folder. Figure 1-2 shows the main elements of Macintosh File Sharing. The File Sharing Extension provides the actual functionality of the AFP server. Three other files—the File Sharing Library, the File Sharing 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.

Figure 1-2 Macintosh File Sharing components



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 File Sharing Library, which allows users to start and to control the File Sharing Extension. The File Sharing Extension communicates with the File Sharing Library primarily by means of server control calls. The File Sharing Extension communicates with the Finder by means of program linking, and with a remote AppleShare client through AFP sessions. The File Sharing Extension also communicates with local volumes and files by means of 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.

File Sharing Library

The File Sharing Library provides the user interface for Macintosh File Sharing. It is an extension of the Finder and resides in the Extensions folder. The File Sharing Library 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 File Sharing control panel.

Based on user interactions, the File Sharing Library communicates with the server primarily by means of server control calls. The File Sharing Extension communicates with users through the File Sharing Library by sending high-level Apple events to display dialog boxes. The File Sharing Library 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. The Finder communicates with the file server by using augmented Macintosh File Manager routines.

File Sharing

This control panel file triggers execution of the appropriate File Sharing Library 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 Sharing Extension by using server control calls. A program that uses server control calls is referred to as a server addition. This manual tells you how to create server additions by using server control calls in your own programs.

Data Files

Both the AppleShare IP 6.2 file server 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 IP 6.2 file server by using the AppleShare IP Web & File 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 IP 6.2 file server, the AppleShare IP Web & File Admin and the AppleShare IP Web & File extension use the Users & Groups Data File. With Macintosh File Sharing, the File Sharing Library and File Sharing Extension 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 Access Privileges folder inside the AppleShare IP Preferences folder (in the Preferences folder) for the AppleShare IP 6.2 file server.

Using Server Control Calls

Table 1-1 lists the server control calls available with the AppleShare IP 6.2 file server.

Table 1-1 Summary of server control calls

Server control call	Supported by Macintosh File Sharing	Function
`SCCancelShutDown` (page 1-34)	Yes	Cancels a shutdown of the file server.
`SCClrCopyProtect` (page 1-35)	No	Disables copy protection for the specified file.
`SCDisconnect` (page 1-36)	No[1]	Disconnects specified users from the file server.
`SCDisconnectVolUsers` (page 1-37)	No	Disconnects specified users from the specified volumes.
`SCGetCacheStats` (page 1-39)	No	Gets cache statistics, such as cache size, utilization, and hits.
`SCGetExpFldr` (page 1-42)	Yes	Gets information about a specific shared volume or folder.
`SCGetExtUserNameRec` (page 1-44)	No	Gets extended information, such as
`SCGetPluginInfo` (page 1-46)	No	Gets information about installed web server plug-ins.
`SCGetPluginMIMEType` (page 1-48)	No	Gets the MIME type for a web server plug-in.
`SCGetServerActivityHistory` (page 1-50)	No	Gets information about server activity, such as minimum, maximum, and average utilization of the server.
`SCGetServerEventProc` (page 1-51)	No	Gets a pointer to the head of the server event handler queue.
`SCGetServerStatus` (page 1-52)	No	Gets information about the number of active sessions, the date of the last modification to the user list, the level of server activity, and the date of the last modification to the volume list.

continued

Table 1-1 Summary of server control calls (continued)

Server control call	Supported by Macintosh File Sharing	Function
`SCGetSetupInfo` (page 1-54)	No	Gets the file server's setup information, including pointer to a `SetupInfoRec` structure, as well as the maximum number of volumes, share points, and concurrent sessions that the server supports.
`SCGetUserMountInfo` (page 1-57)	Yes	Gets information about how a user is using a volume, such as whether the volume is mounted as the owner, the number of files the user has open, and the number files that are open for writing.
`SCGetUserNameRec` (page 1-58)	No	Gets the `UNRecID` ID for a connected user, such as the user's userID, name, log-on time, time of last access, and address from which this user is connected.
`SCInstallServerEventProc` (page 1-60)	No	Installs a server event handler on the file server.
`SCPollServer` (page 1-61)	Yes	Gets information about the server's state, such as its disconnect state, whether or not an error has occurred, and how many seconds until the server shuts down or disconnects a user.
`SCRemoveServerEventProc` (page 1-69)	No	Removes a server event handle from the specified file server.
`SCResetCache` (page 1-70)	No	Flushes the cache.
`SCSendMessage` (page 1-71)	No	Sends a message to specified users.
`SCServerVersion` (page 1-73)	Yes	Gets the name of the file server extension and the server's type and version.
`SCServiceStateInfo` (page 1-74)	No	Gets service state information, such as whether AFP over TCP, HTTP, FTP, and multihoming are enabled.

continued

Table 1-1 Summary of server control calls (continued)

Server control call	Supported by Macintosh File Sharing	Function
`SCSetCopyProtect` (page 1-76)	No	Enables copy protection for the specified file.
`SCSetHistorySampleTime` (page 1-77)	No	Sets the size of each time slice returned by `SCGetActivityHistory`.
`SCSetSetupInfo` (page 1-77)	No	Sets the file server's setup information, such as window visibility and login message.
`SCShutDown` (page 1-78)	Yes	Shuts down the file server.
`SCSleepServer` (page 1-80)	No	Pauses the file server.
`SCStartServer` (page 1-82)	Yes	Starts the file server.
`SCWakeServer` (page 1-82)	No	Starts a file server that has been paused.

[1] Macintosh File Sharing implements the `SCDisconnect` server control call but does not implement the `SCGetUserNameRec` call, so there is no way to obtain a list of users to disconnect.

Determining If Server Control Calls Are Available

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

```
Boolean
TrapAvailable (SInt16 trapNumber, TrapType trapType) {
    /* Check and see if the trap exists. */
    return (NGetTrapAddress (trapNumber, trapType) !=
    GetTrapAddress (_Unimplemented));
} //TrapAvailable

gHasServerDispatch = TrapAvailable (_ServerDispatch, OSTrap);
```

Calling Conventions

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

```
pascal OSErr ServerDispatchSync (SCParamBlockRec *paramBlock);
```

Starting and Stopping the File Server

In Listing 1-1, the `StartStopServer` routine stops or starts the file server, or cancels a shutdown in progress, depending on the current state of the server. To determine the current state of the server, `StartStopServer` calls the `GetServerState` routine, which calls `SCPollServer` (page 1-61) to determine whether the server is running, being shutdown, or not running.

Listing 1-1 Stopping and starting the file server

```
enum {
    kRunningNormally,
    kRunningButShuttingDown,
    kNotRunning
};

OSErr StartStopServer (Boolean startIt, SInt16 howLong) {
    OSErr   err = noErr;
    UInt16  serverState;

    // To decide what to send the server, find out the state it's in.
    // Then make the appropriate judgment.

    err = GetServerState (&serverState);
    if (err == noErr) {
        if (startIt) {
            if (serverState == kRunningNormally) {
                // The file server is already running. Do nothing.
            } else if (serverState == kRunningButShuttingDown) {
                err = CancelShutDown ();
            } else {
```

```
                err = StartServer ();
            } // if
        } else {
            if (serverState != kRunningNormally) {
                // The file server is not running or soon will not be running. Do
                // nothing.
            } else {
                err = ShutDownServer (howLong);
            } // if
        } // if
    } // if
    return err;

} // StartStopServer
```

In Listing 1-2, the `GetServerState` routine calls `SCPollServer` (page 1-61) to determine whether the server is running, being shutdown, or not running.

Listing 1-2 Determining whether the server is running

```
OSErr GetServerState (UInt16* state) {
    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    PollServerParamPtr  pollParam = &serverControl.pollServerParam;
    pollParam->scCode = kSCPollServer;
    pollParam->scSecondsLeft = 0; // For MFS compatibility...
    err = ServerDispatchSync (&serverControl);
    if (pollParam->scServerState == kSCPollRunning) {
        if (pollParam->scDisconnectState == kSCNotDisconnecting) {
            *state = kRunningNormally;
        } else {
            *state = kRunningButShuttingDown;
        } // if
    } else if (pollParam->scServerState == kSCPollStartingUp) {
        *state = kRunningNormally;// will soon be up...
    } else {
        *state = kNotRunning;
    } // if
```

```
    return err;

} // GetServerState
```

The `StartServer` routine shown in Listing 1-3 starts the file server.

Listing 1-3 Starting the file server

```
OSErr StartServer (void) {
    OSErr           err = noErr;
    SCParamBlockRec serverControl;
    StartParamPtr   startParam = &serverControl.startParam;
    startParam->scCode = kSCStartServer;
    startParam->scStartSelect = kSCCurrentlyInstalled;
    startParam->scEventSelect = kSCUseFinderExtension;
    err = ServerDispatchSync (&serverControl);

    return err;

} // StartServer
```

The `ShutDownServer` routine shown in Listing 1-4 stops the file server after a specified period of time.

Listing 1-4 Shutting down the file server

```
OSErr ShutDownServer (SInt16 howLong) {
    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    DisconnectParamPtr  shutDownParam = &serverControl.disconnectParam;
    shutDownParam->scCode = kSCShutDown;
    shutDownParam->scNumMinutes = howLong;
    shutDownParam->scFlags = 0;
    shutDownParam->scMessagePtr = "\pServer is Shutting Down!";
    err = ServerDispatchSync (&serverControl);
```

```
    return err;

} // ShutDownServer
```

The `CancelShutDown` routine shown in Listing 1-5 cancels a shutdown of the file server.

Listing 1-5 Canceling a file server shutdown

```
OSErr CancelShutDown (void) {

    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    DisconnectParamPtr  disconnectParam = &serverControl.disconnectParam;

    disconnectParam->scCode = kSCCancelShutDown;

    err = ServerDispatchSync (&serverControl);

    return err;

} // CancelShutDown
```

Obtaining Status Information about Users, Volumes, and Shared items

This section describes the `SCGetExpFldr` call, which you can use to get information about shared volumes and folders at a specified index position, 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" (page 42) for detailed descriptions of the `SCGetExpFldr` call's return parameters.

The sample code shown in Listing 1-6 gets information about shared volumes and folders.

Listing 1-6 Getting information about shared volumes and folders

```
OSErr GetSharedVolumeInfo (SInt16 vRefNum[], SInt32 dirID[], SInt16 logins[], UInt16 |
                           arraySize) {
    OSErr               err = noErr;
    UInt16              arrayUsed = 0;
    SInt16              curIndex, minIndex, maxIndex;
    SCParamBlockRec     serverControl;
    StandardParamPtr    standardParam = &serverControl.standardParam;

    // Before beginning, determine the minimum and maximum index values for
    // SCGetExpFldr.

    err = GetMinMaxIndexBounds (&minIndex, &maxIndex);
    if (err == noErr) {
        curIndex = minIndex;
        standardParam->scCode = kSCGetExpFldr;
        standardParam->scNamePtr = NULL;// We'll ignore the names...
        while ((arrayUsed < arraySize) && (curIndex <= maxIndex)) {
            err = ServerDispatchSync (&serverControl);
            if (err == noErr) {
                vRefNum[arrayUsed] = standardParam->scVRefNum;
                dirID[arrayUsed] = standardParam->scDirID;
                logins[arrayUsed] = standardParam->scLogins;
                arrayUsed += 1;
            } // if

            curIndex += 1;
            if (err == fnfErr) {
                err = noErr;// Just means the position was empty
            } // if

            if (err != noErr) {
                break;
            } // if

        } // while
    } // if

    return err;
} // GetSharedVolumeInfo
```

The sample code shown in Listing 1-7 gets the range of indices that is valid for calls to `SCGetExpFolder`.

Listing 1-7 Getting the valid range of indices

```
OSErr GetMinMaxIndexBounds (SInt16* minIndex, SInt16* maxIndex) {
    OSErr           err = noErr;
    SCParamBlockRec serverControl;
    SetupInfoRec    setupInfo;
    SetupParamPtr   setupParam = &serverControl.setupParam;

    setupParam->scCode = kSCGetSetupInfo;
    setupParam->scSetupPtr = &setupInfo;
    err = ServerDispatchSync (&serverControl);
    *minIndex = -setupParam->scMaxVolumes;// Volumes are always negative.
    *maxIndex = setupParam->scMaxExpFolders;

    return err;

} // GetMinMaxIndexBounds
```

Sending Messages to Users

The `SendGreetingToAll` routine shown in Listing 1-8 calls `GetNumberOfUsers` (page 1-32) to determine the number of connected users. Then it calls `GetUser` (page 1-33) to get the user name and ID of each connected user. For each connected user, `SendGreetingToAll` calls `SendUserMessage` (page 1-33) to send the message.

Listing 1-8 Sending a message to all connected users

```
OSErr SendGreetingToAll (void) {

    OSErr   err = noErr;
    SInt32  userIndex, numUsers;
    SInt32  userID;
    Str255  userName;
```

```
    // The same message could be sent to all users in one operation, but this
    // routine sends the message one at a time in order to customize the message.

    err = GetNumberOfUsers (&numUsers);
    if (err == noErr) {
        userIndex = 0;
        while (userIndex <= numUsers) {
            err = GetUser (userIndex, userName, &userID);
            if (err == noErr) {
                err = SendUserMessage (userID, userName);
            } else if (err == fnfErr) {
                err = noErr;// User does not exist at this session ID...
            } // if
            if (err != noErr) {
                break;
            } // if
            userIndex += 1;
        } // while
    } // if

    return err;

} // SendGreetingToAll
```

The `GetNumberOfUsers` routine in Listing 1-9 calls `SCGetSetupInfo` to get the number of users who are currently connected.

Listing 1-9 Determining the number of users

```
OSErr GetNumberOfUsers (SInt32* numUsers) {
    OSErr           err = noErr;
    SCParamBlockRec serverControl;
    SetupInfoRec    setupInfo;
    SetupParamPtr   setupParam = &serverControl.setupParam;

    setupParam->scCode = kSCGetSetupInfo;
    setupParam->scSetupPtr = &setupInfo;
    err = ServerDispatchSync (&serverControl);
    *numUsers = setupParam->scCurMaxSessions;
```

```
    return err;

} // GetNumberOfUsers
```

The `GetUser` routine in Listing 1-10 calls `SCGetUserNameRec` to get the name and ID of a user who is connected to the file server.

Listing 1-10 Determining the name and ID of a connected user

```
OSErr GetUser (SInt32 index, StringPtr name, SInt32* userID){
    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    UserInfoParamPtr    userInfoParam = &serverControl.userInfoParam;

    userInfoParam->scCode = kSCGetUserNameRec;
    userInfoParam->scNamePtr = name;
    userInfoParam->scPosition = index;

    err = ServerDispatchSync (&serverControl);
    *userID = userInfoParam->scUNRecID;

    return err;

} // GetUser
```

The `SendUserMessage` routine in Listing 1-11 calls `SCSendMessage` to send the message to the user.

Listing 1-11 Sending the message to a user

```
OSErr SendUserMessage (SInt32 userID, StringPtr name) {

    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    DisconnectParamPtr  messageParam = &serverControl.disconnectParam;
    Str255 message = "\pHello ";
```

```
    messageParam->scCode = kSCSendMessage;
    messageParam->scDiscArrayPtr = &userID;// an array of 1
    messageParam->scArrayCount = 1;
    messageParam->scFlags = 0;
    BlockMoveData (name, &message[StrLength(message) + 1], StrLength (name));
    messageParam->scMessagePtr = message;

    err = ServerDispatchSync (&serverControl);

    return err;

} // SendUserMessage
```

Server Call Reference

This section provides detailed information about each of the AppleShare IP 6.2 file 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. Table 1-2 shows the parameter block for the `SCCancelShutdown` call.

Table 1-2 Parameter block for the SCCancelShutDown call

Parameter	`DisconnectParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word

Field descriptions

`ioResult` Word result value: Result code.

scCode	Word input value: The server control code; always kSCCancelShutDown (0x0003).	

Result Codes			
	noErr	0	No error.
	paramErr	–50	No shutdown or disconnect was in progress.

SCClrCopyProtect

SCClrCopyProtect is called by the AppleShare IP Web & File 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. Table 1-3 shows the parameter block for the SCClrCopyProtect call.

Note
Macintosh File Sharing does not support the SCClrCopyProtect call. ◆

Table 1-3 Parameter block for the SCClrCopyProtect call

Parameter Block				
Parameter Block	StandardParam variant of SCParamBlockRec			
	→	16	ioResult	word
	→	18	scNamePtr	long
	→	22	scVRefNum	word
	→	26	scCode	word
	→	30	scDirID	long

Field descriptions

ioResult	Word result value: Result code.
scCode	Word input value: The server control code; always kSCClrCopyProtect (0x0011).

Result Codes			
	noErr	0	No error.
	paramErr	–50	Server is not running.

Note
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. Table 1-4 shows the parameter block for the SCDisconnect call.

Note
Macintosh File Sharing implements the SCDisconnect server control call but does not implement the SCGetUserNameRec call, so there is no way to obtain a list of users to disconnect. ◆

Table 1-4 Parameter block for the SCDisconnect call

Parameter	DisconnectParam 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

Field descriptions

ioResult	Word result value: Result code.
scDiscArrayPtr	Longword input pointer: Points to the array of 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 kSCDisconnect (0x0004).
scNumMinutes	Word input value: The number of minutes until the users are disconnected, in the range of 0–4094.
scFlags	Word input value: Shutdown flag, as follows: kSCUseMessagePtr The message pointed to by scMessagePtr

should accompany the disconnect. Note that 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.

Result Codes

`noErr`	0	No error.
`kSCAlreadyShuttingDown`	–1	The server is already shutting down.
`kSCAlreadyDisconnecting`	–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
Macintosh File Sharing does not support the disconnect attention message. ◆

Table 1-5 shows the parameter block for the `SCDisconnectVolUsers` call.

Table 1-5 Parameter block for the SCDisconnectVolUsers call

Parameter	DisconnectParam 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

Field descriptions

ioResult	Word result value: Result code.
scDiscArrayPtr	Longword input pointer: Points to the array of 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 kSCDisconnectVolUsers (0x0012).
scNumMinutes	Word input value: The number of minutes until the users are disconnected, in the range of 0–4094.
scFlags	Word input value: Shutdown flag, as follows: kSCUseMessagePtr The message pointed to by scMessagePtr should accompany the disconnect. Note that this feature is not supported by Macintosh File Sharing.
scMessagePtr	Longword input value: A pointer to a buffer containing the message sent to the workstations.

Result Codes			
	noErr	0	No error.
	kSCAlreadyShuttingDown	–1	The server is already shutting down.
	kSCAlreadyDisconnecting	–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.

SCGetCacheStats

SCGetCacheStats returns statistics about the file server cache.

Note
The SCGetCacheStats call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the SCGetCacheStats call. ◆

Table 1-6 shows the parameter block for the SCGetCacheStats call.

Table 1-6 Parameter block for the SCGetCacheStats call

Parameter	GetCacheStatsParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	26	scCode	word
	←→	30	scCacheStatsRecPtr	long
	→	34	scCacheStatsRecSize	long
	←	38	scCacheStatsActSize	long

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `kSCGetCacheStats` (0x0017).

`scCacheStatsRecPtr` Longword input value: A pointer to a `scCacheStatsRec` structure that is to contain the cache statistics.

`scCacheStatsRecSize` Longword input value: The size in bytes of the buffer pointed to by `SCCacheStatsRecPtr`.

`scCacheStatsActSize` Longword output value: The size in bytes of the `SCCachStatsRec` structure containing the cache statistics.

The `scCacheStatsRec` structure is defined as follows:

```
struct SCCacheStatsRec {
    SInt16  csVersion;
    SInt32  csCacheTime;
    SInt32  csRACacheAttempts;
    SInt32  csRACacheHits;
    SInt32  csRACacheTotalEntries;
    SInt32  csRACacheEntriesInUse;
    SInt32  csRACacheEntrySize;
    SInt32  csDirCacheAttempts;
    SInt32  csDirCacheHits;
    SInt32  csDirCacheTotalEntries;
    SInt32  csDirCacheEntriesInUse;
    SInt32  csDirCacheEntrySize;
    SInt32  csIconCacheAttempts;
    SInt32  csIconCacheHits;
    SInt32  csIconCacheTotalEntries;
```

```
    SInt32  csIconCacheEntriesInUse;
    SInt32  csIconCacheEntrySize;
    SInt32  csACtlCacheAttempts;
    SInt32  csACtlCacheHits;
    SInt32  csACtlCacheTotalEntries;
    SInt32  csACtlCacheEntriesInUse;
    SInt32  csACtlCacheEntrySize;
    SInt32  csAUXCacheAttempts;
    SInt32  csAUXCacheHits;
    SInt32  csAUXCacheTotalEntries;
    SInt32  csAUXCacheEntriesInUse;
    SInt32  csAUXCacheEntrySize
};
```

Field descriptions

`csVersion` The version of the `SCCacheStatsRec` structure. For AppleShare IP 6.0, the value of `csVersion` is 3 (`kSCCacheStatsRecVersion`).

`csCacheTime` The time at which this cache information was obtained.

`csRACacheCacheAttempts` The number of attempts to locate a file in the cache.

`csRACacheHits` The number of successful attempts to locate a file in the cache.

`csRACacheTotalEntries` The number of file entries available in the cache.

`csRACacheEntriesInUse` The number of file entries used in the cache.

`csRACacheEntrySize` The size of each file entry in the cache.

`csDirCacheAttempts` The number of attempts to locate a directory in the cache.

`csDirCacheHits` The number of successful attempts to locate a directory in the cache.

`csDirCacheTotalEntries` The number of directory entries available in the cache.

`csDirCacheEntriesInUse` The number of directory entries used in the cache.

`csDirCacheEntrySize` The size of each directory entry in the cache.

`csIconCacheAttempts`
: The number of attempts to locate an icon in the cache.

`csIconCacheHits`
: The number of successful attempts to locate an icon in the cache.

`csIconCacheTotalEntries`
: The number of icon entries in the cache.

`csIconCacheEntriesInUse`
: The number of icon entries used in the cache.

`csIconCacheEntrySize`
: The size of each icon entry in the cache.

`csAUXCacheAttempts`
: Reserved.

`csAUXCacheHits`
: Reserved.

`csAUXCacheTotalEntries`
: Reserved.

`csAUXCacheEntriesInUse`
: Reserved.

`csAUXCacheEntrySize`
: Reserved.

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. ◆

Table 1-7 shows the parameter block for the `SCGetExpFldr` call.

Table 1-7 Parameter block for the SCGetExpFldr call

Parameter	StandardParam 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

Field descriptions

`ioResult` Word result value: Result code.

`scNamePtr` Longword input pointer: Points to the `Str13` where the shared folder's AFP short name will be returned, or must contain NULL. If `scIndex` is negative, then an empty Pascal string (' ') is returned.

`scVRefNum` Word result value: Returns the reference number (`vRefNum`) of the shared folder.

`scLogins` Word result value: Returns the number of people 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 that this value is not returned under Macintosh File Sharing.

`scCode` Word input value: The server control code; always `kSCGetExpFldr` (0x0006).

`scIndex` Word input value: The index into the list of shared folders. Use positive values to get shared folders (what users who are not owners see). Use negative values to get shared volumes (what users who are owners 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.

`scDirID` Longword result value: Returns the directory ID (`dirID`) of the shared folder.

▲ **WARNING**

When `scIndex` is negative, `scNamePtr` must be `NULL`. Otherwise, Macintosh File Sharing writes invalid data into memory. ▲

Result Codes			
	`noErr`	0	No error.
	`fnfErr`	–1	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.

SCGetExtUserName

`SCGetExtUserName` returns information about a user by session index.

Note

The `SCGetExtUserName` call requires AppleShare IP 6.0 or later. ◆

Note

Macintosh File Sharing does not support the `SCGetExtUserName` call. ◆

Table 1-8 shows the parameter block for the `SCGetExtUserName` call.

Table 1-8 Parameter block for the SCGetExtUserName call

Parameter	PluginInfoParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	18	scNamePtr	long
	→	26	scCode	word
	→	28	scPosition	long
	←	32	scUNRecID	long
	←	36	scUserID	long
	→	40	scAttrVersion	word
	→	42	scUserAttrPtr	long

Field descriptions

ioResult	Word result value: Result code.
scNamePtr	Long input value: A pointer to an Str31 where the user name will be returned, or NULL.
scCode	Word input value: The server control code; always kSCGetExtUserName (0x0023).
scPosition	Input value: An index of an active session between 0 and one less than the maximum number of sessions. To get the maximum number of sessions, call SCGetServerStatus (page 1-52).
scUnRecID	Output value: The session ID of the user's session.
scUserID	Output value: The user's user ID.
scAttrVersion	Input value: The version of the UserAttrRec structure pointed to by scUserAttrPtr.
scUserAttrPtr	Input value: Pointer to a UserAttrRec structure in which SCGetExtUserName returns information about the user.

The UserAttrRec structure is defined as follows:

```
struct UserAttrRec {
    SInt32          scLoginTime;
    SInt32          scLastUseTime;
    SInt32          scSocketNum;
    FourCharCode    scProtocolType;
    FourCharCode    scTransportType;
    StringPtr       scSessionNamePtr;
    SInt32          scDisconnectID;
};
```

Field descriptions

`scLoginTime`	The time this login started in seconds since January 1, 1904.
`scLastUseTime`	The time the user was last active in n seconds since January 1, 1904.
`scSocketNum`	The user's AppleTalk or IP address.

`scProtocolType`	The protocol for the session and one of the following constants:
	`kSCSessionAFP = 'afp ';`
	`kSCSessionHTTP = 'http';`
	`kSCSessionFTP = 'ftp ';`
	`kSCSessionSMB = 'smb ';`

`scTransportType`	The transport protocol for the session and one of the following constants:
	`kSCTransportATP = 'atp ';`
	`kSCTransportTCP = 'tcp ';`

`scSessionNamePtr`	A textual description of the session type (can be `NULL`).
`scDisconnectID`	`TRUE` if the user is in the process of being disconnected via the countdown timer.

Result Codes			
	`noErr`	0	No error.
	`paramErr`	–50	The server is not running, `attrVersion` is not known, or `scPosition` is out of range.
	`fnfErr`	–43	There is no valid user at this position.

SCGetPluginInfo

`SCGetPluginInfo` returns information about web server plug-ins.

Note
The `SCGetPluginInfo` call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the `SCGetPluginInfo` call. ◆

Table 1-9 shows the parameter block for the `SCGetPluginInfo` call.

Table 1-9 Parameter block for the SCGetPluginInfo call

Parameter	`PluginInfoParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word
	→	28	`version`	word
	→	30	`index`	long
	←	34	`error`	word
	←	36	`name`	32-byte array
	←	68	`versionStr`	12-byte array
	←	80	`adminURL`	256-byte array
	←	336	`plugInRef`	long
	←	340	`isLast`	word

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `kSCGetPluginInfo` (0x0029).

`version` Word input value: The version of `SCGetPluginInfo` that your application supports. For AppleShare IP 6.0, the value of `version` should be zero.

`index` Longword input value: A value specifying the plug-in for which information is to be returned. Your application should start with `index` set to `kSCPlugInInfoParamVersion` (0x0000) and increment index until `SCGetPluginInfo` returns a value of `isLast` that is `TRUE`.

`error` An error code that can be one of the following values:

```
enum {
 kSCPlugInNoErr= 0,
 kSCPlugInWrongVersionErr= 1,
 kSCPlugInBadIndexErr= 2,
 kSCPlugInPlugInsNotLoadedErr = 3,
 kSCPlugInBadPlugInRefErr= 4
};
```

name	Output C string: The name of the plug-in.
versionStr	Output C string: The plug-in's version string.
adminURL	Output C string: The universal resource locator (URL) of the supporting plug-in.
plugInRef	Output word value: A value that identifies a plug-in. You can use plugInRef to make an SCGetPluginMIMEType call.
isLast	Output word value: TRUE if SCGetPluginInfo returned information about the last plug-in; otherwise, the value of isLast is FALSE.

Result Codes

noErr	0	No error.
paramErr	–50	The server is not running.

SCGetPluginMIMEType

SCGetPluginMIMEType returns information about the MIME types that the web server's plug-ins support.

Note
The SCGetPluginMIMEType call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the SCGetGetPluginMIMEType call. ◆

Table 1-10 shows the parameter block for the SCGetPluginMIMEType call.

Table 1-10 Parameter block for the SCGetPluginMIMEType call

Parameter	`PluginMIMETypeParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word
	→	28	`version`	word
	→	30	`plugInRef`	long
	→	34	`index`	word
	←	36	`error`	byte
	←	38	`mimetype`	80-byte array
	←	118	`suffix`	32-byte array
	←	170	`typeCode`	long
	←	174	`creatorCode`	long
	←	178	`isLast`	word

Field descriptions

`ioResult` Word result value: Result code.

`reserved` Reserved input value.

`reserved2` Reserved input value.

`reserved3` Reserved input value.

`scCode` Word input value: The server control code; always `kSCGetPluginMIMEType` (0x002A).

`version` Word input value: The version of `SCGetPluginInfo` that your application supports. For AppleShare IP 6.0, the value of `version` should be zero.

`plugInRef` Input word value: A value returned by `SCGetPluginInfo` that identifies the plug-in for which MIME type information is to be obtained.

`index` Longword input value: A value specifying the MIME type information that is to be returned. Your application should start with `index` set to zero and increment index until `SCGetPluginMIMEType` returns a value of `isLast` that is `TRUE`.

`error` Output An error code that can be one of the following values:

```
enum {
 kSCPlugInNoErr= 0,
 kSCPlugInWrongVersionErr= 1,
 kSCPlugInBadIndexErr= 2,
 kSCPlugInPlugInsNotLoadedErr = 3,
 kSCPlugInBadPlugInRefErr= 4
};
```

mimeType	Output C string: A value that represents one of the MIME types the plug-in supports.
suffix	Output C string: A value that represents the suffix for the MIME type contained by mimeType.
typeCode	Output long: The plug-in's type code.
creatorCode	Output long: The plug-in's creator code.
isLast	Output word value: TRUE if SCGetPluginMIMEType returned information about the last MIME type that the plug-in supports; otherwise, the value of isLast is FALSE.

Result Codes

noErr	0	No error.
paramErr	–50	Server not running.

SCGetServerActivityHistory

SCGetServerActivityHistory returns information about file server activity, including the minimum, maximum, and average utilization of the file server.

Note
The SCGetServerActivityHistory call requires AppleShare IP 6.0 or later. ◆

By default, the AppleShare IP 6.2 file server takes a sample every 10 milliseconds. Information is returned in a server history record that contains 1024 samples, which is enough to store about 10.25 seconds of history data at the default sample rate.

Note
Macintosh File Sharing does not support the SCGetServerActivityHistory call. ◆

Table 1-11 shows the parameter block for the SCGetServerActivityHistory call.

Table 1-11 Parameter block for the SCGetServerActivityHistory call

Parameter Block				
Parameter	`GetHistoryParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	←	18	`scHistory`	long
	→	22	`numDataPointsRequested`	word
	→	26	`scCode`	word

Field descriptions

`ioResult` Word result value: Result code.

`scHistory` Output long: A pointer to a server history record that contains the history data.

`numDataPointsRequested` The number of data points to return in the server history record.

`scCode` Word input value: The server control code; always `kSCGetServerActivityHistory` (0x002C).

Result Codes

`noErr`	0	No error.

SCGetServerEventProc

`SCGetServerEventProc` returns the head of the server event handler queue.

Note
Macintosh File Sharing does not support the `SCGetServerEventProc` call. ◆

Table 1-12 shows the parameter block for the `SCGetServerEventProc` call.

Table 1-12 Parameter block for the SCGetServerEventProc call

Parameter	`ServerEventParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	←	18	`scSEQEntryPtr`	long
	→	26	`scCode`	word

Field descriptions

`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)thePB.scSEQEntryPtr)->qHead`.

`scCode` Word input value: The server control code; always `kSCGetServerEventProc` (0x000D).

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

SCGetServerStatus

`SCGetServerStatus` returns server status information.

Note
Macintosh File Sharing does not support the `SCGetServerStatus` call. ◆

Table 1-13 shows the parameter block for the `SCGetServerStatus` call.

Table 1-13 Parameter block for the SCGetServerStatus call

Parameter	`StatusParam` 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

Field descriptions

`ioResult` Word result value: Result code.

`scNamePtr` 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 `kSCGetServerEventProc` (0x000A).

`scServerFlags` Word result value: Obsolete.

`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 in a SetupInfoRec structure. The SetupInfoRec structure is defined as follows:

```
struct SetupInfoRec {
    SInt16      siVersion;
    SInt16      siFlags;
    SInt16      siMaxLogins;
    SInt16      siSrvrUsageLimit;
    Point       siVolInfoLocation;
    Boolean     siVolInfoVisible;
    Boolean     siReserved1;
    Point       siUserInfoLocation;
    Boolean     siUserInfoVisible;
    Boolean     siReserved2;
    SInt16      siShutDownMins;
    SInt16      siCacheControl;         /* No longer used */
    SInt16      siVolParmsStepSize;     /* No longer used */
    SInt16      siVolParmsIncrement;    /* No longer used */
    SInt16      siVolParmsFirstDelay;   /* No longer used */
    SInt16      siVolParmsMaxDelay;     /* No longer used */
    SInt32      siRACacheFileBufSize;   /* No longer used */
    SInt32      siRACacheSize;          /* No longer used */
    SInt16      siDirCacheMaxWidth;     /* No longer used */
    SInt32      siDirCacheSize;         /* No longer used */
    SInt32      siIconCacheSize;        /* No longer used */
    SInt32      siBTMemReservedFromCache;
    SInt16      siSpare[1];             /* Reserved */
    Str198      siLoginMsg;
};
typedef struct SetupInfoRecSetupInfoRec;
```

Field descriptions

`siVersion`	The version of the `SetupInfoRec` structure. For AppleShare IP 6.0, the value of `siVersion` is 3 (`kSCSetupRecordVersion`).
`siFlags`	Reserved. Set to zero.
`siMaxLogins`	The maximum number of logins for which the server is configured.
`siSrvrUsageLimit`	The maximum amount of the computer's processing power that is allocated to the file server.
`siVolInfoLocation`	The location of the Volume Info window.
`siVolInfoVisible`	`TRUE` if the Volume Info window is visible; `FALSE` if the Volume Info window is not visible.
`siReserved1`	Reserved.
`siUserInfoLocation`	The location of the Connected Users window.
`siUserInfoVisible`	`TRUE` if the Connected Users window is visible; `FALSE` if the Connected Users window is not visible.
`siReserved2`	Reserved.
`siShutDownMins`	The number of minutes that is used by default for shutting down the file server.
`siCacheControl`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siVolParmsStepSize`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siVolParmsIncrement`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siVolParmsFirstDelay`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siVolParmsMaxDelay`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siRACacheFileBufSize`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siRACacheSize`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
`siDirCacheMaxWidth`	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).

siDirCacheSize	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
siIconCacheSize	Obsolete. To obtain this information, see "SCGetCacheStats" (page 39).
siBTMemReservedFromCache	The amount of memory that is reserved for applications other than the file server.
siSpare	Reserved.
siLoginMsg	A string containing the message that is displayed when users log on.

Table 1-14 shows the parameter block for the SCGetServerSetupInfo call.

Table 1-14 Parameter block for the SCGetSetupInfo Call

Parameter SetupParam variant of SCParamBlockRec

Block				
	←	16	ioResult	word
	→	18	scSetupPtr	long
	←	26	scMaxVolumes	word
	←	28	scMaxExpFolders	word
	→	30	scCode	word
	←	32	scCurMaxSessions	word

Field descriptions

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 NULL.
scMaxVolumes	Word result value: Returns the maximum number of volumes supported by the server. Note that 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 that 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 kSCGetSetupInfo (0x0007).
scCurMaxSessions	Word result value: Returns the maximum number of logins currently allowed. Note that this value is not returned under Macintosh File Sharing.

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 acVRefNum is positive), these values are for that shared folder only. For a real volume (that is, if the value of acVRefNum is negative), these values represent totals for all shared folders on the volume.

Table 1-15 shows the parameter block for the SCGetUserMountInfo call.

Table 1-15 Parameter block for the SCGetUserMountInfo call

Parameter	VolMountedParam 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

Field descriptions

ioResult	Word result value: Result code.
scVRefNum	Longword input value: The volume specification or shared folder specification.

scCode	Word input value: The server control code; always kSCGetUserMountInfo (0x0014).
scFilesOpen	Word result value: Returns the total number of files the user has open on the volume or shared folder.
scWriteableFiles	Word input 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	Word 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 by being an its owner.

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
Macintosh File Sharing does not support the SCGetUserNameRec call. ◆

Table 1-16 shows the parameter block for the SCGetUserNameRec call.

Table 1-16 Parameter block for the SCGetUserNameRec call

Parameter	`UserInfoParam` 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

Field descriptions

`ioResult` Word result value: Result code.

`scNamePtr` Longword result pointer: Points to a `Str31` where the user name will be copied, or must contain NULL.

`scCode` Word input value: The server control code; always `kSCGetUserNameRec` (0x0013).

`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 access the server.

`scSocketNum` Longword result value: Returns the AppleTalk network address or the IP address this user is connected from. The value is returned in an `AddrBlock` record.

Result Codes	noErr	0	No error.
	nsvErr	–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.

SCInstallServerEventProc

SCInstallServerEventProc installs a server event object in the server event handler queue. For sample code, see “Sample Server Event Handler Code” (page 94).

Note
Macintosh File Sharing does not support the SCInstallServerEventProc call. ◆

Table 1-17 shows the parameter block for the SCInstallServerEventProc call.

Table 1-17 Parameter block for the SCInstallServerEventProc call

Parameter Block	ServerEventParam variant of SCParamBlockRec			
	←	16	ioResult	word
	→	18	scServerEventQEntry	long
	→	26	scCode	word

Field descriptions

ioResult — Word result value: Result code.

scServerEventQEntry — 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 kSCInstallServerEventProc (0x000B).

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. Table 1-18 shows the parameter block for the SCPollServer call.

Table 1-18 Parameter block for the SCPollServer call

Parameter	PollServerParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	26	scCode	word
	←	28	scServerState	word
	←	30	scDisconnectState	word
	←	32	scServerError	word
	←	34	scSecondsLeft	long

Field descriptions

ioResult	Word result value: Result code.
scCode	Word input value: The server control code; always kSCPollServer (0x0005).

`scServerState` Word result value: The state of the server, as follows:

`kSCDisconnectWithin29Secs`	0–29 seconds before shutdown; Network Setup message says “Less than a minute.
`kSCDisconnect30To89Secs`	30–89 seconds before shutdown; Network Setup message says “About a minute.”
0x0002–0x0FFE	(`scServerState`*60) – 30 to (`scServerState`*60) + 29 seconds before shutdown; Network Setup message says “About *scServerState* minutes.”
`kSCPollRunning`	Server running normally.
`kSCPollStartingUp`	Server is in the process of starting up.
`kSCPollJustDisabled`	Server was just disabled and there was no startup error.
`kSCPollDisabledErr`	Server is disabled and there is an “SE” error in `scServerError`.
`kSCPollSleeping`	Server is temporarily disabled. Note that this result is not returned by Macintosh File Sharing.

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

`kSCDisconnectWithin29Secs`	0–29 seconds before disconnect; Network Setup message says “Less than a minute.
`kSCDisconnect30To89Secs`	30–89 seconds before disconnect; Network Setup message says “About a minute.”
0x0002–0x0FFE	(`scDisconnectState`*60) – 30 to (`scDisconnectState`*60) + 29 seconds before disconnect; Network Setup message says “About *scDisconnectState* minutes.”

	kSCNotDisconnecting	Server not disconnecting some user or group of users.
	kSCJustDisabled	Server was just disabled and there was no startup error.
	kSCDisabledErr	Server is disabled and there is an “SE” error in scServerError.
	kSCSleeping	Server is temporarily disabled. Note that this result is not returned by Macintosh File Sharing.

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

kSCModernMemMgrOffErr	The Modern Memory Manager is not enabled.
kSCNoThreadLibraryErr	The Thread Manager could not be found.
kSCServiceNotInstalledErr	The specified service is not installed.
kSCInsuffMFMemErr	There was not enough memory in the Process Manager’s heap for the server to start up.
kSCCantRegNameErr	The file server’s name could not be registered on the AppleTalk network.
kSCCantFindExtnFolderErr	The file server could not be started because the Extensions folder could not be found.
kSCUnExATalkErr	An unexpected AppleTalk error occurred.
kSCNoMachineNameErr	The computer on which the file server is installed does not have a name.

kSCCantFindFSExtnErr	The file server could not start up because the AppleShare IP Web & File Server extension or the File Sharing Extension could not be found.
kSCATalkOffErr	AppleTalk is turned off.
kSCNoInitRunErr	The AppleShare IP Web & File Server extension or File Sharing Extension is not installed in the System Folder.
kSCInsuffAppMemErr	There was not enough memory for the file server to start up.
kSCBadConfigErr	The file server encountered a problem with the current configuration.
kSCNoDTOnStartupErr	The desktop database on the startup volume could not be opened.
kSCDupNameErr	Duplicate-name error occurred when the server was registering. Choose another name for this computer.
kSCBadFileBufParmsErr	Obsolete.
kSCNeedRootUserErr	Administrator privileges are required to complete the specified action.
kSCBadSerialNumErr	The specified AppleShare IP serial number is invalid.

kSCSysTooOldErr	The System file is too old for the AppleShare IP 6.2 file server.
kSCDupSerialNumberErr	An other computer is running AppleShare using the same serial number as this computer.
kSCVMOnErr	Obsolete.
kSCBadInitErr	An inconsistency between components has been detected; reinstall AppleShare IP 6.2.
kSCOpenTransportInstallErr	The version of Open Transport installed on this computer is incompatible with this version of AppleShare IP.
kSCNoAgentLibErr	The AppleShare Registry Library could not be found.
kSCInvalidAgentErr	The AppleShare Registry Agent is not running or is not responding.
kSCAgentServerObjErr	Bad server object type.
kSCCorruptedMimeTypesErr	The defined set of MIME types that the server supports is invalid.
kSCAgentGenesisErr	The AppleShare Registry could not start up.
kSCAlreadyShuttingDown	The server is already shutting down.

kSCAlreadyDisconnecting	The sever is already scheduled to disconnect users.
kSCDeletedPDSErr	The PDS file could not be found.
kSCContainsExpFolderErr	A sharepoint contains another sharepoint.
kSCCantPrepareVolumeErr	The specified volume could not be shared.
kSCTooManyExpFoldersErr	Too many folders are configured for sharing.
kSCFixedPDSErr	The AppleShare PDS file was damaged, but the server has repaired it.
kSCExpFolderNamConfErr	Two or more share points have the same name.
kSCNoExportFolderErr	No folders are being shared.
kSCInsideExpFolderErr	A share point is contained within another share point.
kSCInsideTrashErr	A share point is in the Trash.
kSCVolNameConflictErr	Two or more volumes have the same name.
kSCCacheReducedErr	Obsolete.
kSCBadIPConfigErr	The TCP/IP control panel is configured incorrectly.
kSCBadAccessPrivRecErr	The access privilege record is invalid.
kSCBadMimeTypeFileErr	The file that contains the MIME types that the server supports is invalid.
kSCAFPGenErr	Generic AFP error.

kSCAFPTCPGenErr	Generic TCP over AFP error.
kSCAFPTCPMemErr	A TCP over AFP memory error occurred.
kSCAFPTCPPortInUseErr	The port used by TCP over AFP is already in use.
kSCFTPGenErr	Generic FTP error.
kSCFTPPortInUseErr	The port used by FTP is already in use.
kSCFTPNotAvailErr	FTP is not enabled.
kSCFTPMemErr	An FTP memory error occurred.
kSCHTTPGenErr	Generic HTTP error.
kSCHTTPPortInUseErr	The port used by HTTP is already in use.
kSCHTTPFolderErr	The folder that contains the home page cannot be found.
kSCHTTPFileErr	The file that contains the home page cannot be found or contains an error.
kSCHTTPMemErr	An HTTP memory error occurred.
kSCHTTPNoMimeTypesErr	The web server does not support any MIME types.
kSCHTTPNoDefaultMimeErr	The default MIME type is not defined.
kSCPluginDirNotFoundErr	The plug-in folder cannot be found.
kSCPluginMemFullErr	The memory allocated for plug-ins is full.
kSCPluginPreProcNotFoundErr	A plug-in preprocessor could not be found.

kSCPluginPostProcNotFoundErr	A plug-in postprocessor could not be found.
kSCErrorPluginNotFoundErr	The plug-in specified for handling errors could not be found.
kSCPluginNotPreProcessorErr	The plug-in specified for preprocessing requests could not be found.
kSCPluginNotPostProcessorErr	The plug-in specified for postprocessing requests could not be found.
kSCPluginMemPoolFullErr	The memory pool for plug-ins could not be allocated.
kSCPluginOutOfMemoryErr	The plug-in failed to load because it was out of memory.
kSCCorruptedMimeTypesErr	The list of MIME types that the server supports is invalid.
kSCPlugInLoggingErr	A plug-in logging error occurred.
kSCPlugInTypeConflictErr	Two or more plug-ins support the same MIME type.
kSCPlugInCannotRegisterErr	A plug-in failed to register itself.
kSCPlugInMemSmallErr	The requested amount of memory for plug-ins was not available, so a smaller amount was allocated.

	kSCWebAdminNetworkErr	A low-level networking error occurred when the server tried to allocate resources for the web administration port. The port may be in use by another program.
	kSCSMBGenErr	Generic SMB error.
	kSCSMBPortInUseErr	The port used by SBM is already in use.
	kSCSMBMemErr	An SMB memory error occurred.

scSecondsLeft Longword result value: Returns the number of seconds left before the shutdown or disconnect. Zero is returned if no shutdown or disconnect is in progress. This value is undefined if the server is disabled (not running). Note that 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
Macintosh File Sharing does not support the SCRemoveServerEventProc call. ◆

Table 1-19 shows the parameter block for the SCRemoveServerEventProc call.

Table 1-19 Parameter block for the SCRemoveServerEventProc call

Parameter	ServerEventParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	18	scSEQEntryPtr	long
	→	26	scCode	word

Field descriptions

ioResult — Word result value: Result code.

scSEQEntryPtr — Longword input pointer: Points to the ServerEventQEntry server event object to be removed from the server event handler queue.

scCode — Word input value: The server control code; always kSCRemoveServerEventProc (0x000C).

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 in the server event handler queue.

SCResetCache

SCResetCache flushes the file server cache.

Note
The SCResetCache call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the SCResetCache call. ◆

Table 1-20 shows the parameter block for the SCResetCache call.

Table 1-20 Parameter block for the SCResetCache call

Parameter	ResetCacheParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	26	scCode	word
	→	28	bitmap	word

Field descriptions

ioResult — Word result value: Result code.

scCode — Word input value: The server control code; always kSCResetCache (0x001F).

bitmap — A bitmask consisting of a combination of the following constants:

kSCShrinkCache, performs the specified action when combined with one or more of the constants that follow.
kSCResetFileCache, resets the file cache of read ahead and write behind data.
kSCResetCNodeCache, resets the cache of directory information.
kSCResetDTCache, resets the desktop cache containing permission information.
kSCShrinkAllCaches, resets all caches to their initial sizes.

Result Codes	noErr	0	No error.

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
Macintosh File Sharing does not support the SCSendMessage call. ◆

Table 1-21 shows the parameter block for the SCSendMessage call.

Table 1-21 Parameter block for the SCSendMessage call

Parameter	DisconnectParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	18	scSEQEntryPtr	long
	→	22	scArrayCount	word
	→	26	scCode	word
	→	30	scFlags	word
	→	32	scMessagePtr	word

Field descriptions

`ioResult` Word result value: Result code.

`scSEQEntryPtr` 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 `kSCSendMessage` (0x0009).

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

`kSCUseMessagePtr`	There is a message pointed to by `scMessageErr`.

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

Result Codes			
	`noErr`	0	No error.
	`kSCAlreadyShuttingDown`	–1	The server is already shutting down.
	`kSCAlreadyDisconnecting`	–2	The server is already disconnecting.
	`paramErr`	–50	The server is not running or a `UnRecID` is invalid.

SCServerVersion

`SCServerVersion` returns the name of the file server extension and the server's type and version.

Note
Macintosh File Sharing does not return a valid value for `scServerVersion` if the server is not running. ◆

Table 1-22 shows the parameter block for the `SCServerVersion` call.

Table 1-22 Parameter block for the SCServerVersion call

Parameter	`VersionParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	←	18	`scExtNamePtr`	long
	→	26	`scCode`	word
	←	28	`scServerType`	word
	←	30	`scServerVersion`	word

Field descriptions

`ioResult` Word result value: Result code.

`scExtNamePtr` Longword result pointer: Points to a `Str31` where the server application name (the name of the INIT) will be returned, or must contain NULL.

`scCode` Word input value: The server control code; always `kSCServerVersion` (0x000E).

`scServerType` Word result value: Returns the server type, as follows:

0x0000	Macintosh File Sharing (`kSCMFSServerType`).
0x0001	AppleShare file server (`kSCAFSServerType`).

scServerVersion	Word input value: Returns the server version, as follows.	
	0x0600	The value returned by AppleShare IP 6.0. AppleShare IP 6.0.1 will return 0x0601, and so on.
	0x0052	The value returned by AppleShare IP 5.0, 5.0.1, 5.0.2, and 5.0.3.

Result Codes	noErr	0	No error.

SCServiceStateInfo

SCServiceStateInfo returns information about the services that are enabled on the file server.

Note
The SCServiceStateInfo call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the SCGetServerStateInfo call. ◆

Table 1-23 shows the parameter block for the SCServiceStateInfo call.

Table 1-23 Parameter block for the SCServiceStateInfo call

Parameter	`ServiceStateParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word
	←	28	`afpTCPState`	word
	←	30	`httpState`	word
	←	32	`ftpState`	word
	←	34	`multihoming`	word
	←	36	`srvrUsageLimit`	word

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `kSCServiceStateInfo` (0x0026).

`afpTCPState` Output word value: `TRUE` if AFP over the Transmission Control Protocol (TCP) is enabled on the server; `FALSE` if AFP over TCP/IP is not enabled.

`httpState` Output word value: `TRUE` if the Hypertext Transfer Protocol (HTTP) is enabled on the server; `FALSE` if HTTP is not enabled.

`ftpState` Output word value: `TRUE` if the File Transmission Protocol (FTP) is enabled on the server; `FALSE` if FTP is not enabled.

`multihoming` Output word value: `TRUE` if multihoming is enabled; `FALSE` if multihoming is not enabled.

`srvrUsageLimit` Output word value: A value indicating the amount of the computer's processing power that is allocated to the file server.

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

SCSetCopyProtect

SCSetCopyProtect is called by the AppleShare IP Web & File 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
Macintosh File Sharing does not support the SCSetCopyProtect call. ◆

Table 1-24 shows the parameter block for the SCSetCopyProtect call.

Table 1-24 Parameter block for the SCSetCopyProtect call

Parameter	StandardParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	18	scNamePtr	long
	→	22	scVRefNum	word
	→	26	scCode	word
	→	30	scDirID	long

Field descriptions

ioResult	Word result value: Result code.
scNamePtr	Longword input pointer: Points to the file name.
scVRefNum	Word input value: The volume specification
scCode	Word input value: The server control code; always kSCSetCopyProtect (0x0010).
scDirID	Longword input value: The parent directory ID.

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

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

SCSetHistorySampleTime

SCSetHistorySampleTime sets the history sample time.

Note
The SCSetHistorySampleTime call requires AppleShare IP 6.0 or later. ◆

Note
Macintosh File Sharing does not support the SCSetHistorySampleTime call. ◆

Table 1-25 shows the parameter block for the SCSetHistorySampleTime call.

Table 1-25 Parameter block for the SCSetHistorySampleTime call

Parameter	SetHistoryParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	24	historySampleTime	word
	→	26	scCode	word

Field descriptions

ioResult	Word result value: Result code.
historySampleTime	Size of time slice to be returned by SCGetServerActivityHistory (page 1-50).
scCode	Word input value; always kSCSetHistorySampleTime (0x002B).

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 structure) do not take effect until the next time the AppleShare IP file server application starts up:

- siVolInfoLocation, which defines the location of Volume Info window.
- siVolInfoVisible, which defines whether the Volume Info window is visible.

- siUserInfoLocation, which defines the location of the Connected Users window.
- siUserInfoVisible, which defines whether the Connected Users window is visible.

The SetupInfoRec structure is described in "SCGetSetupInfo" (page 54).

Note
Macintosh File Sharing does not support the SCSetSetupInfo call. ◆

Table 1-26 shows the parameter block for the SCSetSetupInfo call.

Table 1-26 Parameter block for the SCSetSetupInfo call

Parameter	SetupParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	18	scSetupPtr	long
	→	26	scCode	word

Field descriptions

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 (0x0008).

Result Codes	noErr	0	No error.
	paramErr	–50	The server is not running, scSetupPtr is NULL, 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 IP Web & File Server application automatically quits if the AppleShare IP file server is shut down with the `SCShutDown` call. ▲

Table 1-27 shows the parameter block for the `SCShutDown` call.

Table 1-27 Parameter block for the SCShutDown call

Parameter	`DisconnectParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word
	→	28	`scNumMinutes`	word
	→	30	`scSFlags`	word
	→	32	`scMessagePtr`	long

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `ksSCShutDown` (0x0002).

`scNumMinutes` Word input value: The number of minutes until server shutdown, in the range 0–4094.

`scSFlags` Word input value: Shutdown flag, as follows:

`kSCUseMessagePtr`	The message pointed to by `scMessagePtr` should accompany the disconnect. Note that 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 that this feature is not supported by Macintosh File Sharing.

Result Codes			
	noErr	0	No error.
	kSCAlreadyShuttingDown	–1	The server is already shutting down.
	kSCAlreadyDisconnecting	–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 IP 6.2 file server 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). You might want to put a file server to sleep before switching networks or temporarily turning off AppleTalk.

SCSleepServer fails if the server is starting up.

Note
Macintosh File Sharing does not support the SCSleepServer call. ◆

Table 1-28 shows the parameter block for the SCSleepServer call.

Table 1-28 Parameter block for the SCSleepServer call

Parameter	DisconnectParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	26	scCode	word
	→	28	scNumMinutes	word
	→	30	scSFlags	word
	→	32	scMessagePtr	long

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `SCSleepServer` (0x0016).

`scNumMinutes` Word input value: The number of minutes until server sleep, in the range 0–4094.

`scSFlags` Word input value: Shutdown flag, as follows:

`kSCUseMessagePtr`	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.
`kSCAlreadyShuttingDown`	–1	The server is already shutting down.
`kSCAlreadyDisconnecting`	–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.

Table 1-29 shows the parameter block for the SCStartServer call.

Table 1-29 Parameter block for the SCStartServer call

Parameter	StartParam variant of SCParamBlockRec			
Block	←	16	ioResult	word
	→	26	scCode	word
	→	28	scNumMinutes	word
	→	30	scSFlags	word

Field descriptions

ioResult — Word result value: Result code.

scCode — Word input value: The server control code; always kSCStartServer (0x0021).

scNumMinutes — 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 or Macintosh File Sharing.

scEventSelect — Word input value: Always kFinderExtn.

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

Result Codes

noErr	0	No error.
paramErr	–50	The file server is not running.

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

SCWakeServer

SCWakeServer starts the file server.

Note
Macintosh File Sharing does not support the `SCWakeServer` call. ◆

Table 1-30 shows the parameter block for the `SCWakeServer` call.

Table 1-30 Parameter block for the SCWakeServer call

Parameter	`StartParam` variant of `SCParamBlockRec`			
Block	←	16	`ioResult`	word
	→	26	`scCode`	word

Field descriptions

`ioResult` Word result value: Result code.

`scCode` Word input value: The server control code; always `kSCWakeServer` (0x0015).

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

Note
Other errors from waking the server—such as `fnfErr` and `memFullErr`—may also be returned. ◆

CHAPTER 2

Server Event Handling

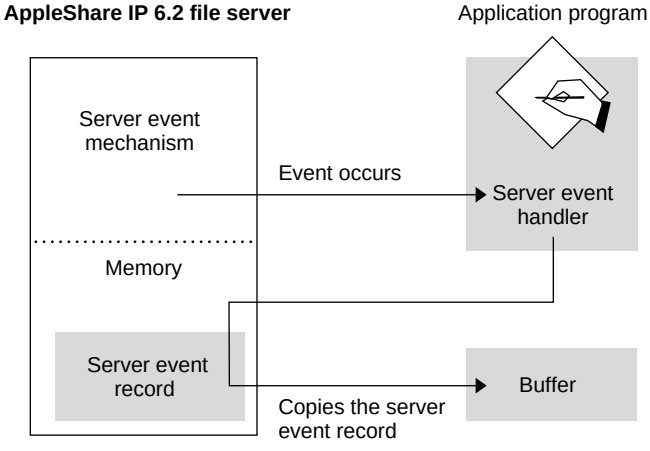
Contents

This 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 IP 6.2 file 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. Figure 2-1 shows how the server event mechanism works.

Figure 2-1 The Server Event Mechanism



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 "SCInstallServerEventProc" in Chapter 1, "Server Control Calls." 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: Networking*.)

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 `serverEventMask`, `serverControlMask`, and the `afpCommandMask` array as appropriate to see if the handler is interested in the event that just occurred. If it is, the server calls the handler, passing pointers to the `ServerEventQEntry` 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.

If an AFP event occurs and you want to handle that event, set the bit corresponding to the AFP event in `afpCommandMask` and then set either the `kSCStartAFPRequestEvt` bit or the `kSCSendAFPResponseEvt` bit in `serverEventMask` to specify whether you want the handler to be called before or after the event is processed, respectively.

If a server control call is made that you want to intercept, set the bit corresponding to the command number in `serverControlMask` and set `kSCServerControlCallEvt` in `serverEventMask`. To intercept other events, set the corresponding bit in `serverEventMask`.

Server Event Queue Entry

The server event queue entry structure is defined as follows:

```
struct ServerEventQEntry {
    ServerEventQEntry*      next;
    SInt16                  queuetype;
    ServerEventHandlerUPP   callBack;
    SInt32                  serverEventMask;
    SInt32                  afpCommandMask[2];
    SInt32                  serverControlMask;
};
```

Field descriptions

`next` Filled in by the server.

`queuetype` Filled in by the server.

`callBack` The event handler itself.

`serverEventMask` Server event mask. A combination of one or more of the following values:

```
kSCStartAFPRequestEvt = 0 /* AFP before processing */
kSCSendAFPResponseEvt = 1 /* AFP after processing */
kSCServerControlCallEvt = 4 /* SC call made */
kSCServerBusyEvt = 2 /* Server-generated events */
kSCServerShutdownEvt = 3
kSCShareEvt = 5
kSCUnShareEvt = 6
kSCSetDirAccessEvt = 7
kSCServerNameChangeEvt = 8
```

```
                    kSCVolumePrepEvt = 9
                    kSCVolumeUnmountEvt             = 10
                    kSCServerStartupEvt             = 11
                    kSCSessionTornDownEvt           = 12
                    kSCOutOfSequenceEvt             = 13
                    kSCWksClosedSessionEvt          = 14
                    kSCSessionTimedOutEvt           = 15
                    kSCSrvrClosedSessionEvt         = 16
```

`afpCommandMask` Bit positions corresponding to AFP command codes. For an example, see Listing 2-7 in *Inside AppleTalk*. This field is only relevant if `kSCStartAFPRequestEvt`, `kSCSendAFPResponseEvt`, or both, is set in `serverEventMask`.

`serverControlMask` Bit position corresponding to server control calls. This field is only relevant if `kSCServerControlCallEvt` is set in `serverEventMask`.

Note
`kSCStartAFPRequestEvt` and `kSCSendAFPResponseEvt` must be set in combination with `afpCommandMask`. `kSCServerControlCallEvt` must be set in combination with `serverControlMask`. ◆

Server Event Record

AppleShare 5.0.3 and earlier uses server event records to store information about server events. The `ServerEventRecord` structure defines the server event record:

```
struct ServerEventRecord {
    SInt32      eventNumber;
    UInt32      serverTimeInSeconds;
    SInt16      result;
    SInt16      bufferSize;
    char        buffer[48];
    Str31       nameStr;
    SInt16      afpCommand;
    SInt32      sessionID;
    SInt32      userID;
    Str31       userName;
    SInt16      vRefNum;
```

```
    SInt32      dirID;
    UserAddress addr;
};
typedef struct ServerEventRecord    ServerEventRecord;
```

Field descriptions

`eventNumber` The server event that occurred.

`serverTimeInSeconds` The time that the event occurred in standard Macintosh date/time form

`result` Any error code associated with the event.

`bufferSize` The size in bytes of the valid data in `buffer`.

`buffer` The AFP packet, `SCParamBlockRec`, `HParamBlockRec`, or the new server name (up to a maximum of 48 bytes)

`nameStr` The name of the file or directory upon which the operation is being performed (if applicable).

`afpCommand` The AFP command.

`sessionID` The session ID.

`userID` The user's user ID.

`userName` The user name of the user performing this operation (registered users only).

`vRefNum` The reference number of the volume upon which this operation was performed (if applicable).

`dirID` The directory ID of the volume upon which this operation was performed (if applicable).

`addr` The socket address of this user (provided in address block (`AddrBlock`) format (net number:node ID:socket number) or an IP address if the user connects via TCP/IP.

Extended Server Event Record

AppleShare IP 6.2 and later uses extended server event records to store information about server events. AppleShare 5 and earlier uses server event records to store information about server events. The `ExtendedServerEventRecord` structure defines the extended server event record:

```
struct ExtendedServerEventRecord {
    SInt32          eventNumber;
    UInt32          serverTimeInSeconds;
    SInt16          result;
    SInt16          bufferSize;
    char            buffer[48];
    Str31           nameStr;
    SInt16          afpCommand;
    SInt32          sessionID;
    SInt32          userID;
    Str31           userName;
    SInt16          vRefNum;
    SInt32          dirID;
    UserAddress     addr;
    FourCharCode    transportType;
    UInt32          annexVersion;
};
```

Field descriptions

`eventNumber` The server event that occurred.

`serverTimeInSeconds` The time that the event occurred in standard Macintosh date/time form

`result` Any error code associated with the event.

`bufferSize` The size in bytes of the valid data in `buffer`.

`buffer` The AFP packet, `SCParamBlockRec`, `HParamBlockRec`, or the new server name (up to a maximum of 48 bytes)

`nameStr` The name of the file or directory upon which the operation is being performed (if applicable).

`afpCommand` The AFP command.

`sessionID` The session ID.

`userID` The user's user ID.

`userName` The user name of the user performing this operation (registered users only).

`vRefNum` The reference number of the volume upon which this operation was performed (if applicable).

`dirID` The directory ID of the volume upon which this operation was performed (if applicable).

addr	The socket address of this user (provided in address block (AddrBlock) format (net number:node ID:socket number) or an IP address if the user connects over TCP/IP.
transportType	The transport type (kSCTransportATP for AppleTalk or kSCTransportTCP for TCP/IP).
annexVersion	The version of the record. For AppleShare IP 6.0, the value of annexVersion is kServerEventAnnexVersion6.

Server Event Definitions

Table 2-1 lists server event constants and codes.

Table 2-1 Server Event Definitions

Constant	Code	Meaning
kSCStartAFPRequestEvt	0	The server received an AFP request.
kSCSendAFPResponseEvt	1	The server sent a reply to an AFP request.
kSCServerBusyEvt	2	The server is too busy to respond; for example, the server may not have a socket to allocate for a connection request.
kSCServerShutdownEvt	3	The server is being shut down.
kSCServerControlCallEvt	4	The server received a server control call.
kSCShareEvt	5	A new share point is being shared.
kSCUnShareEvt	6	A previously shared share-point is no longer being shared.
kSCSetDirAccessEvt	7	New access privileges have been applied to a shared folder.
kSCServerNameChangeEvt	8	The server's name has been changed.
kSCVolumePrepEvt	9	A sharable volume has been prepared; for example, a CD-ROM disc has been inserted.
kSCVolumeUnmountEvt	10	A volume was unmounted.
kSCServerStartupEvt	11	The server was started.
kSCSessionTornDownEvt	12	A user disconnected or was disconnected.

Table 2-1 Server Event Definitions (continued)

Constant	Code	Meaning
kSCOutOfSequenceEvt	13	A client sent a duplicate request or sent an unexpected sequence number.
kSCWksClosedSessionEvt	14	A user disconnected.
kSCSessionTimedOutEvt	15	A session timed out.
kSCSrvrClosedSessionEvt	16	A session closed but has not yet disconnected.
kSCExtendedServerEvtRec	31	Indicates that your event handler can assume that it will receive extended server event records (page 2-91) from the server (as opposed to server event records).

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.

In Listing 2-1, the `InstallRemoveEventHandler` routine installs an event handle queue entry on the server and removes the event handler when it is no longer needed.

Listing 2-1 Installing and removing a server event handler

```
OSErr InstallOrRemoveEventHandler (Ptr seqEntry, Boolean install);
    OSErr               err = noErr;
    SCParamBlockRec     serverControl;
    ServerEventParamPtr serverEventParam = &serverControl.serverEventParam;

    if (install) {
        serverEventParam->scCode = kSCInstallServerEventProc;
    } else {
        serverEventParam->scCode = kSCRemoveServerEventProc;
    } // if
    serverEventParam->scSEQEntryPtr = seqEntry;

    err = ServerDispatchSync (&serverControl);

    return err;

} // InstallOrRemoveEventHandler
```

In Listing 2-2, the `OurServerEventRecord` structure adds some fields before `ExtendedServerEventRecord` structure so that you can manipulate the structure with the standard OS queue manipulation routines. Similarly, the `OurServerEventQEntry` structure adds two fields after the `ServerEventQEntry` structure so it can be accessed from the event handler.

Listing 2-2 Preparing structures for use with queue manipulation routines

```
typedef struct {
    QElemPtr                    qLink;  // Make OS queue-compatible
    SInt16                      qType;
```

```
    ExtendedServerEventRecord   eventRec;
} OurServerEventRecord;

typedef struct {
    ServerEventQEntry   queueEntry;     // Actual queue entry
    QHdr                freeQ;          // List of free OurServerEventRecord
    QHdr                usedQ;          // List of used OurServerEventRecord
} OurServerEventQEntry;
```

In Listing 2-3, `InitServerEventQueueData` creates a queue entry for receiving events.

Listing 2-3 Creating a queue entry for receiving events

```
OSErr InitServerEventQueueData (OurServerEventQEntry* eventQueueEntry,
                            OurServerEventRecord toQueue[], UInt32 numEventRecords) {
    OSErr                           err = noErr;
    QHdr                            emptyQueueInit = { 0, NULL, NULL };
    static ServerEventHandlerUPP    ourCallBack = NULL;

    // Create the callback...
    if (ourCallBack == NULL) {
        ourCallBack = NewServerEventHandlerProc (ServerEventHandler);
    } // if

    eventQueueEntry->queueEntry.callBack = ourCallBack;

    // Initially, clear all flags.
    eventQueueEntry->queueEntry.serverEventMask = 0;
    eventQueueEntry->queueEntry.afpCommandMask[0] = 0;
    eventQueueEntry->queueEntry.afpCommandMask[1] = 0;
    eventQueueEntry->queueEntry.serverControlMask = 0;

    // Caller returned a block of OurServerEventRecords; push on the free queue.
    eventQueueEntry->freeQ = emptyQueueInit;
    eventQueueEntry->usedQ = emptyQueueInit;

    while (numEventRecords > 0) {
        numEventRecords -= 1;
        Enqueue ((QElemPtr) &toQueue[numEventRecords], &eventQueueEntry->freeQ);
```

```
    } // while

    return err;

} // InitServerEventQueueData
```

In Listing 2-4, `ServerEventHandler` receives events and puts them on a queue for the application to process later.

Listing 2-4 Receiving and queuing events

```
pascal void ServerEventHandler (OurServerEventQEntry* mainEntry,
                                ExtendedServerEventRecord* event) {

    OSErr                   err = noErr;
    OurServerEventRecord*   newEntry;

    // If there is free space in the queue, get it; if there is not, purge the oldest
    // item in the used queue (you may want to behave differently, such as purging
    // items that are of less interest, etc.)

    newEntry = (OurServerEventRecord*) mainEntry->freeQ.qHead;
    if (newEntry != NULL) {
        err = Dequeue ((QElemPtr) newEntry, &mainEntry->freeQ);
    } else {
        newEntry = (OurServerEventRecord*) mainEntry->usedQ.qHead;
        err = Dequeue ((QElemPtr) newEntry, &mainEntry->usedQ);
    } // if

    // Now you have an entry; stuff the event record into it and requeue it on the
    // "used" side.
    if (err == noErr) {
        newEntry->eventRec = *event;
        Enqueue ((QElemPtr) newEntry, &mainEntry->usedQ);
    } // if

} // ServerEventHandler
```

The `SetEventFlag` routine shown in Listing 2-5 determines which server events the handler will receive.

Listing 2-5 Determining which server events to receive

```
void SetEventFlag (OurServerEventQEntry* mainEntry, UInt32 whichEvent, Boolean onOff) {
    UInt32  maskValue = 0x1 << whichEvent;
    if (onOff) {
        mainEntry->queueEntry.serverEventMask |= maskValue;
    } else {
        mainEntry->queueEntry.serverEventMask &= ~maskValue;
    } // if

} // SetEventFlag
```

The `SetControlFlag` routine shown in Listing 2-6 determines which server control calls the handler will receive.

Listing 2-6 Determining which server control calls to receive

```
// SetControlFlag determines what server _control_ calls a handler will receive.
void SetControlFlag (OurServerEventQEntry* mainEntry, UInt32 whichEvent,
                     Boolean onOff) {
    UInt32  maskValue = 0x1 << whichEvent;
    if (onOff) {
        mainEntry->queueEntry.serverControlMask |= maskValue;
    } else {
        mainEntry->queueEntry.serverControlMask &= ~maskValue;
    } // if
} // SetControlFlag
```

The `SetAFPFlag` routine shown in Listing 2-7 determines which AFP calls the handler will receive.

Note
The handler can intercept only calls that are in the range of 1 to 64. ◆

Listing 2-7 Determining which AFP calls to receive

```
void SetAFPFlag (OurServerEventQEntry* mainEntry, UInt32 whichEvent, Boolean inDo,
                Boolean inReply, Boolean onOff) {
    UInt32  maskValue0 = 0;
    UInt32  maskValue1 = 0;

    // Special case of AddIcon gets remapped to bit 0.
    if (whichEvent == afpAddIcon) {
        whichEvent = 0;
    } // if

    if (whichEvent >= 32) {
        maskValue0 = 1 << (whichEvent % 32);
    } else {
        maskValue1 = 1 << whichEvent;
    } // if

    if (onOff) {
        mainEntry->queueEntry.afpCommandMask[0] |= maskValue0;
        mainEntry->queueEntry.afpCommandMask[1] |= maskValue1;
    } else {
        mainEntry->queueEntry.afpCommandMask[0] &= ~maskValue0;
        mainEntry->queueEntry.afpCommandMask[1] &= ~maskValue1;
    } // if

    // Set the appropriate Event flag(s) so this actually gets called.
    if (inDo) {
        SetEventFlag (mainEntry, kSCStartAFPRequestEvt, onOff);
    } // if

    if (inReply) {
        SetEventFlag (mainEntry, kSCSendAFPResponseEvt, onOff);
    } // if

} // SetAFPFlag
```

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.

In Listing 2-8, the `ProcessQueuedEvents` routine goes through the events that have been queued by the `ServerEventHandler` routine shown in Listing 2-4 (page 97) and processes them (in this case, it simply beeps).

Listing 2-8 Processing server events

```
void ProcessQueuedEvents (OurServerEventQEntry* mainEntry) {
    OSErr                   err = noErr;
    OurServerEventRecord*   nextEntry;
    nextEntry = (OurServerEventRecord*) mainEntry->usedQ.qHead;
    if (nextEntry != NULL) {
        err = Dequeue ((QElemPtr) nextEntry, &mainEntry->usedQ);
        if (err == noErr) {
            SysBeep (0);
            Enqueue ((QElemPtr) nextEntry, &mainEntry->freeQ);
        } // if
    } // if

} // ProcessQueuedEvents
```

Appendixes

APPENDIX A

Macintosh File Sharing Server Control Calls

Macintosh File Sharing supports a subset of the AppleShare IP 5.0 file 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 IP 6.2 file server 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 IP 6.2 file server. 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 code in the section "Obtaining Status Information about Users, Volumes, and Shared items" (page 29) in Chapter 1, "Server Control Calls."

`SCNamePtr` must be `NULL` when `scIndex` is negative. Otherwise, Macintosh File Sharing writes garbage into memory. See the comments in the sample function code listed in the section "Obtaining Status Information about Users, Volumes, and Shared items" (page 29) in Chapter 1, "Server Control Calls."

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 5.)

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

- `siVolInfoVisible`
- `siUserInfoLocation`
- `siUserInfoVisible`
- `siShutDownMins`
- `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.

Interface Files

The AppleShareFileServerControl.h, AppleShareFileServerControl.p, and AppleShareFileServerControl.a files contain all of the definitions for the server control calls used to control Macintosh File Sharing and the AppleShare IP 6.2 file server, for C, Pascal, and assembly language, respectively. This appendix presents portions of the AppleShareFileServerControl.h file.

Server Control Constants

The server control constants are defined as follows:

```
enum {
    kSCStartServer              = 0, /* Use StartParam variant */
    kSCShutDown                 = 2, /* Use DisconnectParam variant */
    kSCCancelShutDown           = 3, /* Use DisconnectParam variant */
    kSCDisconnect               = 4, /* Use DisconnectParam variant */
    kSCPollServer               = 5, /* Use PollServerParam variant */
    kSCGetExpFldr               = 6, /* Use StandardParam variant */
    kSCGetSetupInfo             = 7, /* Use SetupParam variant */
    kSCSetSetupInfo             = 8, /* Use SetupParam variant */
    kSCSendMessage              = 9, /* Use DisconnectParam variant */
    kSCGetServerStatus          = 10, /* Use StatusParam variant */
    kSCInstallServerEventProc   = 11, /* Use ServerEventParam variant */
    kSCRemoveServerEventProc    = 12, /* Use ServerEventParam variant */
    kSCGetServerEventProc       = 13, /* Use ServerEventParam variant */
    kSCServerVersion            = 14, /* Use VersionParam variant */
    kSCSetCopyProtect           = 16, /* Use StandardParam variant */
    kSCClrCopyProtect           = 17, /* Use StandardParam variant */
    kSCDisconnectVolUsers       = 18, /* Use DisconnectParam variant */
    kSCGetUserNameRec           = 19, /* Use UserInfoParam variant */
    kSCGetUserMountInfo         = 20, /* Use VolMountedParam variant */
    kSCWakeServer               = 21, /* Use StartParam variant */
    kSCSleepServer              = 22, /* Use DisconnectParam variant */
    kSCGetCacheStats            = 23, /* Use GetCacheStatsParam variant */
    kSCResetCache               = 31, /* Use ResetCacheParam variant */
    kSCServiceStateInfo         = 38, /* Use ServiceStateParam variant */
```

```
    kSCGetPlugInInfo            = 41, /* Use PlugInInfoParam variant */
    kSCGetPlugInMimeType        = 42, /* Use PlugInMimeTypeParam variant */
    kSCSetHistorySampleTime     = 43, /* Use SetHistoryParam variant */
    kSCGetServerActivityHistory = 44 /* Use GetHistoryParam variant */
};

/* scFlags bits and masks for DisconnectParam */
    kSCUseMessagePtr = 1;
{some constants for SCStartServer}
    kSCCurrentlyInstalled = 0;  {use currently installed server}
    kSCUseFinderExtension = 0;  {use the Finder extension}

enum {
    kSCMFSServerType= 0x0000,   /* Macintosh File Sharing */
    kSCAFSServerType= 0x0001    /* AppleShare/AppleShare IP File Server*/
};

enum {
    kSCPollRunning          = -1,
    kSCPollStartingUp       = -2,
    kSCPollJustDisabled     = -3,
    kSCPollDisabledErr      = -4,
    kSCPollSleeping         = -5
};

    /* Disconnect state responses returned by SCPollServer */
enum {
    kSCNotDisconnecting         = -1,
    kSCDisconnectWithin29Secs   = 0,
    kSCDisconnect30To89Secs     = 1     /* Any other value is the number
                                           of minutes remaining, rounded. */
};

/* Server errors returned by SCPollServer */
enum {
    kSCModernMemMgrOffErr           = 1, /* Must run with Modern Memory Manager */
    kSCNoThreadLibraryErr           = 2,
    kSCServiceNotInstalledErr       = 3,
    kSCInsuffMFMemErr               = 4,
    kSCCantRegNameErr               = 5,
    kSCCantFindExtnFolderErr        = 6,
```

```
kSCUnExATalkErr                 = 7,
kSCNoMachineNameErr             = 8,
kSCCantFindFSExtnErr            = 9,
kSCATalkOffErr                  = 10,
kSCNoInitRunErr                 = 12,
kSCInsuffAppMemErr              = 14,
kSCBadConfigErr                 = 15,
kSCNoDTOnStartupErr             = 16,
kSCDupNameErr                   = 17,
kSCBadFileBufParmsErr           = 19,
kSCNeedRootUserErr              = 20,
/* The range 21-28 are reserved for future use by Apple Computer. */
kSCBadSerialNumErr              = 29,
kSCSysTooOldErr                 = 34,
kSCDupSerialNumberErr           = 36,   /* NBP duplicate serial number detected */
kSCVMOnErr                      = 37,   /* On the server, virtual memory is on */
kSCNoPPCErr                     = 38,   /* Server only runs on a PPC machine */
kSCBadInitErr                   = 39,
kSCOpenTransportInstallErr      = 40,   /* Incompatible version of Open
                                           Transport */
kSCNoAgentLibErr                = 41,   /* No ASRLib */
kSCNoAgentSessionErr            = 42,   /* Could not open an agent session */
kSCInvalidAgentErr              = 43,   /* No agent or a problem with the agent */
kSCAgentServerObjErr            = 44,   /* Bad server object type */
kSCCorruptedMimeTypesErr        = 45,
kSCAgentGenesisErr              = 46,
kSCAlreadyShuttingDown          = -1,
kSCAlreadyDisconnecting         = -2,
kSCDeletedPDSErr                = -2,
kSCContainsExpFolderErr         = -3,
kSCCantPrepareVolumeErr         = -4,
kSCTooManyExpFoldersErr         = -5,
kSCFixedPDSErr                  = -6,
kSCExpFolderNamConfErr          = -7,
kSCNoExportFolderErr            = -8,
kSCInsideExpFolderErr           = -9,
kSCInsideTrashErr               = -10,
kSCVolNameConflictErr           = -11,
kSCCacheReducedErr              = -12,
kSCBadIPConfigErr               = -20,
kSCBadAccessPrivRecErr          = -21,
```

```
    kSCBadMimeTypeFileErr           = -22,
/* -100 to -199 are AFP errors */
    kSCAFPGenErr                    = -100,
    kSCAFPTCPGenErr                 = -150,
    kSCAFPTCPMemErr                 = -151,
    kSCAFPTCPPortInUseErr           = -152,
/* -200 to -299 are FTP errors */
    kSCFTPGenErr                    = -200,
    kSCFTPPortInUseErr              = -201,
    kSCFTPNotAvailErr               = -202,
    kSCFTPMemErr                    = -203,
/* -300 to -399 are Web errors */
    kSCHTTPGenErr                   = -300,
    kSCHTTPPortInUseErr             = -302,
    kSCHTTPFolderErr                = -303,
    kSCHTTPFileErr                  = -304,
    kSCHTTPMemErr                   = -305,
    kSCHTTPNoMimeTypesErr           = -306,
    kSCHTTPNoDefaultMimeErr         = -307,
    kSCPluginDirNotFoundErr         = -308,
    kSCPluginMemFullErr             = -309,
    kSCPluginPreProcNotFoundErr     = -310,
    kSCPluginPostProcNotFoundErr    = -311,
    kSCErrorPluginNotFoundErr       = -312,
    kSCPluginNotPreProcessorErr     = -313,
    kSCPluginNotPostProcessorErr    = -314,
    kSCPluginMemPoolFullErr         = -315,
    kSCPluginOutOfMemoryErr         = -316,
    kSCCorruptedMimeTypesErr        = -317,
    kSCPlugInLoggingErr             = -318,
    kSCPlugInTypeConflictErr        = -319,
    kSCPlugInCannotRegisterErr      = -320,
    kSCPlugInMemSmallErr            = -321,
    kSCWebAdminNetworkErr           = -330,
/* -400 to -499 are SMB errors  */
    kSCSMBGenErr                    = -400,
    kSCSMBPortInUseErr              = -402,
    kSCSMBMemErr                    = -405
};
```

Server Control Parameter Blocks

The server control parameter blocks are defined as follows:

```
union SCParamBlockRec {
    StartParam          startParam;
    DisconnectParam     disconnectParam;
    PollServerParam     pollServerParam;
    StandardParam       standardParam;
    SetupParam          setupParam;
    StatusParam         statusParam;
    ServerEventParam    serverEventParam;
    VersionParam        versionParam;
    UserInfoParam       userInfoParam;
    VolMountedParam     volMountedParam;
    GetCacheStatsParam  getCacheStatsParam;
    ResetCacheParam     resetCacheParam;
    ExtUserInfoParam    extUserInfoParam;
    ServiceStateParam   serviceStateParam;
    PlugInInfoParam     plugInInfoParam;
    PlugInMimeTypeParam plugInMimeTypeParam;
    SetHistoryParam     setHistoryParam;
    GetHistoryParam     getHistoryParam;
};
typedef union SCParamBlockRec SCParamBlockRec;

struct StartParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          scStartSelect;
    SInt16          scEventSelect;
    SInt32          scWhere;
    SInt32          scReceiverID;
```

```
    SInt32          scDataType;
    SInt32scStartOptions;
};
typedef struct StartParam StartParam;
typedef StartParam *StartParamPtr;

struct DisconnectParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    SInt32 *        scDiscArrayPtr;
    SInt16          scArrayCount;
    SInt16          reserved;
    SInt16          scCode;
    SInt16          scNumMinutes;
    SInt16          scFlags;
    StringPtr       scMessagePtr;
};
typedef struct DisconnectParam DisconnectParam;
typedef DisconnectParam *DisconnectParamPtr;

struct PollServerParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          scServerState;
    SInt16          scDisconnectState;
    SInt16          scServerError;
    SInt32          scSecondsLeft;
```

```
};
typedef struct PollServerParam PollServerParam;
typedef PollServerParam *PollServerParamPtr;

struct StandardParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    StringPtr       scNamePtr;
    SInt16          scVRefNum;
    SInt16          scLogins;
    SInt16          scCode;
    SInt16          scIndex;
    SInt32          scDirID;
};
typedef struct StandardParam StandardParam;
typedef StandardParam *StandardParamPtr;

struct SetupParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap*/
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    SetupInfoPtr    scSetupPtr;
    SInt16          scMaxVolumes;
    SInt16          scMaxExpFolders;
    SInt16          scCode;
    SInt16          scCurMaxSessions;
};
typedef struct SetupParam SetupParam;
typedef SetupParam *SetupParamPtr;

struct StatusParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
```

```
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    StringPtr       scNamePtr;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          scServerFlags;
    SInt16          scNumSessions;
    SInt32          scUserListModDate;
    SInt16          scActivity;
    SInt32          scVolListModDate;
};
typedef struct StatusParam StatusParam;
typedef StatusParam *StatusParamPtr;

struct ServerEventParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to*/
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    Ptr             scSEQEntryPtr;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
};
typedef struct ServerEventParam ServerEventParam;
typedef ServerEventParam *ServerEventParamPtr;

struct VersionParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code*/
    StringPtr       scExtNamePtr;
    SInt16          reserved2;
    SInt16          reserved3;
```

```
    SInt16          scCode;
    SInt16          scServerType;
    SInt16          scServerVersion;
};
typedef struct VersionParam VersionParam;
typedef VersionParam *VersionParamPtr;

struct UserInfoParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    StringPtr       scNamePtr;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt32          scPosition;
    SInt32          scUNRecID;
    SInt32          scUserID;
    SInt32          scLoginTime;
    SInt32          scLastUseTime;
    SInt32          scSocketNum;
};
typedef struct UserInfoParam UserInfoParam;
typedef UserInfoParam *UserInfoParamPtr;

struct VolMountedParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap*/
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    Ptr             reserved;
    SInt16          scVRefNum;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          scFilesOpen;
    SInt16          scWriteableFiles;
```

```
    SInt32          scUNRecID;
    Boolean         scMounted;
    Boolean         scMountedAsOwner;
};
typedef struct VolMountedParam VolMountedParam;
typedef VolMountedParam *VolMountedParamPtr;

struct GetCacheStatsParam {
    QElemPtr            qLink;          /* queue link in header*/
    SInt16              qType;          /* type byte for safety check*/
    SInt16              ioTrap;         /* FS: the Trap*/
    Ptr                 ioCmdAddr;      /* FS: address to dispatch to*/
    SCCompletionUPP     ioCompletion;   /* Completion routine addr (0 for sync calls)*/
    OSErr               ioResult;       /* result code*/
    Ptr                 reserved;
    SInt16              reserved2;
    SInt16              reserved3;
    SInt16              scCode;
    SInt16              reserved4;
    SCCacheStatsRecPtr  scCacheStatsPtr;
    SInt16              scCacheStatsReqSize;
    SInt16              scCacheStatsActSize;
};
typedef struct GetCacheStatsParam GetCacheStatsParam;
typedef GetCacheStatsParam *GetCacheStatsParamPtr;

struct ResetCacheParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code*/
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          bitmap;
};
typedef struct ResetCacheParam ResetCacheParam;
typedef ResetCacheParam *ResetCacheParamPtr;
```

```
struct ExtUserInfoParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    StringPtr       scNamePtr;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt32          scPosition;
    SInt32          scUNRecID;
    SInt32          scUserID;
    SInt16          attrVersion;
    UserAttrPtr     scUserAttrPtr;
};
typedef struct ExtUserInfoParam ExtUserInfoParam;
typedef ExtUserInfoParam *ExtUserInfoParamPtr;

struct ServiceStateParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code*/
    StringPtr       reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          afpTCPState;
    SInt16          httpState;
    SInt16          ftpState;
    SInt16          multiHoming;
    SInt16          srvrUsageLimit;
};
typedef struct ServiceStateParam ServiceStateParam;
typedef ServiceStateParam *ServiceStateParamPtr;
```

```
struct PlugInInfoParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code*/
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          version;
    SInt32          index;
    SInt16          error;
    char            name[32];
    char            versionStr[12];
    char            adminURL[256];
    UInt32          plugInAttributes;
    SInt32          plugInRef;
    SInt16          isLast;
};
typedef struct PlugInInfoParam PlugInInfoParam;
typedef PlugInInfoParam *PlugInInfoParamPtr;

struct PlugInMimeTypeParam {
    QElemPtr        qLink;          /* Queue link in header */
    SInt16          qType;          /* Type byte for safety check */
    SInt16          ioTrap;         /* FS: the Trap */
    Ptr             ioCmdAddr;      /* FS: address to dispatch to */
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code */
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          reserved3;
    SInt16          scCode;
    SInt16          version;
    SInt32          plugInRef;
    SInt32          index;
    SInt16          error;
    char            mimetype[80];
    char            suffix[32];
```

```
    OSType          typeCode;
    OSType          creatorCode;
    SInt16          isLast;
};
typedef struct PlugInMimeTypeParam PlugInMimeTypeParam;
typedef PlugInMimeTypeParam *PlugInMimeTypeParamPtr;

struct SetHistoryParam {
    QElemPtr        qLink;          /* Queue link in header*/
    SInt16          qType;          /* Type byte for safety check*/
    SInt16          ioTrap;         /* FS: the Trap*/
    Ptr             ioCmdAddr;      /* FS: address to dispatch to*/
    SCCompletionUPP ioCompletion;   /* Completion routine addr (0 for sync calls) */
    OSErr           ioResult;       /* Result code*/
    SInt32          reserved;
    SInt16          reserved2;
    SInt16          historySampleTime;
    SInt16          scCode;
};
typedef struct SetHistoryParam SetHistoryParam;
typedef SetHistoryParam *SetHistoryParamPtr;
```

Server Control Records

The server control records are defined as follows:z

```
struct SetupInfoRec {
    SInt16      siVersion;
    SInt16      siFlags;
    SInt16      siMaxLogins;
    SInt16      siSrvrUsageLimit;
    Point       siVolInfoLocation;
    Boolean     siVolInfoVisible;
    Boolean     siReserved1;
    Point       siUserInfoLocation;
    Boolean     siUserInfoVisible;
    Boolean     siReserved2;
    SInt16      siShutDownMins;
    SInt16      siCacheControl;         /* No longer used */
    SInt16      siVolParmsStepSize;
    SInt16      siVolParmsIncrement;
```

```
    SInt16      siVolParmsFirstDelay;
    SInt16      siVolParmsMaxDelay;
    SInt32      siRACacheFileBufSize;   /* No longer used */
    SInt32      siRACacheSize;          /* No longer used */
    SInt16      siDirCacheMaxWidth;     /* No longer used */
    SInt32      siDirCacheSize;         /* No longer used */
    SInt32      siIconCacheSize;        /* No longer used */
    SInt32      siBTMemReservedFromCache;
    SInt16      siSpare[1];             /* Reserved */
    Str198      siLoginMsg;
};
typedef struct SetupInfoRec SetupInfoRec;
typedef SetupInfoRec *SetupInfoPtr;

struct SCCacheStatsRec {
    SInt16      csVersion;
    SInt32      csCacheTime;
    SInt32      csRACacheAttempts;      /* File cache */
    SInt32      csRACacheHits;
    SInt32      csRACacheTotalEntries;
    SInt32      csRACacheEntriesInUse;
    SInt32      csRACacheEntrySize;
    SInt32      csDirCacheAttempts;     /* Directory cache */
    SInt32      csDirCacheHits;
    SInt32      csDirCacheTotalEntries;
    SInt32      csDirCacheEntriesInUse;
    SInt32      csDirCacheEntrySize;
    SInt32      csIconCacheAttempts;    /* Desktop cache */
    SInt32      csIconCacheHits;
    SInt32      csIconCacheTotalEntries;
    SInt32      csIconCacheEntriesInUse;
    SInt32      csIconCacheEntrySize;
    SInt32      csACtlCacheAttempts;    /* PDS info, part of directory cache */
    SInt32      csACtlCacheHits;
    SInt32      csACtlCacheTotalEntries;
    SInt32      csACtlCacheEntriesInUse;
    SInt32      csACtlCacheEntrySize;
    SInt32      csAUXCacheAttempts;     /* Not used */
    SInt32      csAUXCacheHits;
    SInt32      csAUXCacheTotalEntries;
    SInt32      csAUXCacheEntriesInUse;
```

```
    SInt32      csAUXCacheEntrySize;
                /*  New fields for version 3 record*/
    SInt32      csEnumCacheAttempts;
    SInt32      csEnumCacheHits;
    SInt32      csEnumCacheTotalEntries;
    SInt32      csEnumCacheEntriesInUse;
    SInt32      csEnumCacheEntrySize;
    SInt32      csMaxFBUsed;
    SInt32      csSkipPrsAttempts;
    SInt32      csSkipPrsHits;
};
typedef struct SCCacheStatsRec SCCacheStatsRec;
typedef SCCacheStatsRec *SCCacheStatsRecPtr;

/* Used in extended user call if attrVersion is kOldUserAttrRecVersion */
struct OldUserAttrRec {
    SInt32  scLoginTime;
    SInt32  scLastUseTime;
    SInt32  scSocketNum;
    SInt16  scConnectionType;
    SInt16  scDisconnectID;
};
typedef struct OldUserAttrRec OldUserAttrRec;
typedef OldUserAttrRec *OldUserAttrPtr;

/* Used in extended user call if attrVersion is kUserAttrRecVersion */
struct UserAttrRec {
    SInt32          scLoginTime;
    SInt32          scLastUseTime;
    SInt32          scSocketNum;
    FourCharCode    scProtocolType;     /* The session protocol, i.e. AFP, FTP, SMB */
    FourCharCode    scTransportType;    /* The transport, i.e. ATP, TCP/IP */
    StringPtr       scSessionNamePtr;   /* Str63 */
    SInt32          scDisconnectID;
};
typedef struct UserAttrRec UserAttrRec;
typedef UserAttrRec *UserAttrPtr;

struct HistoryData {
    UInt8   dpMin;
    UInt8   dpMax;
```

```
    UInt8   dpAverage;
    UInt8   filler;
};
typedef struct HistoryDataHistoryData;

struct ServerHistoryRec {
    UInt32      historySyncCount;
    UInt32      historyLastSample;
    UInt16      historySampleTime;
    UInt16      numDataPoints;
    HistoryData dataPoint[1024];
};

typedef struct ServerHistoryRec ServerHistoryRec;
typedef ServerHistoryRec *ServerHistoryPtr;
```

Server Control Routine

```
/* C */
pascal OSErr ServerDispatchSync (SCParamBlockRec* paramBlock);

; Assembly
$A094   ServerDispatch;
```

Server Events

Server Event Constants

```
/* Bit names for the serverEventMask field of ServerEventQEntry; event numbers
   returned in ServerEventRecord.
*/

enum {
    kSCStartAFPRequestEvt   = 0,
    kSCSendAFPResponseEvt   = 1,
    kSCServerBusyEvt        = 2,
    kSCServerShutdownEvt    = 3,
```

```
    kSCServerControlCallEvt = 4,
    kSCShareEvt             = 5,
    kSCUnShareEvt           = 6,
    kSCSetDirAccessEvt      = 7,
    kSCServerNameChangeEvt  = 8,
    kSCVolumePrepEvt        = 9,
    kSCVolumeUnmountEvt     = 10,
    kSCServerStartupEvt     = 11,
    kSCSessionTornDownEvt   = 12,
    kSCOutOfSequenceEvt     = 13,
    kSCWksClosedSessionEvt  = 14,
    kSCSessionTimedOutEvt   = 15,
    kSCSrvrClosedSessionEvt = 16,
    kSCExtendedServerEvtRec = 31
};

/* Maximum size of the Buffer in the ServerEventRecord... */
enum {
kBufferMax          = 48
};
```

Server Event Data Types

```
struct ServerEventQEntry {
    ServerEventQEntry*      next;
    SInt16                  queuetype;
    ServerEventHandlerUPP   callBack;
    SInt32                  serverEventMask;
    SInt32                  afpCommandMask[2];
    SInt32                  serverControlMask;
};

struct ServerEventRecord {
    SInt32      eventNumber;
    UInt32      serverTimeInSeconds;
    SInt16      result;
    SInt16      bufferSize;
    char        buffer[48];
    Str31       nameStr;
    SInt16      afpCommand;
```

```
    SInt32      sessionID;
    SInt32      userID;
    Str31       userName;
    SInt16      vRefNum;
    SInt32      dirID;
    UserAddress addr;
};
typedef struct ServerEventRecord ServerEventRecord;

/* For annexVersion field; set by server to indicate version of record... */
enum {
    kServerEventAnnexVersion6 = 0x06000000
};

struct ExtendedServerEventRecord {
    SInt32          eventNumber;
    UInt32          serverTimeInSeconds;
    SInt16          result;
    SInt16          bufferSize;
    char            buffer[48];
    Str31           nameStr;
    SInt16          afpCommand;
    SInt32          sessionID;
    SInt32          userID;
    Str31           userName;
    SInt16          vRefNum;
    SInt32          dirID;
    UserAddress     addr;
    FourCharCode    transportType;
    UInt32          annexVersion;
};
typedef struct ExtendedServerEventRecord ExtendedServerEventRecord;
```

Application-Defined Routine

```
pascal void
ServerEventHandlerProcPtr (ServerEventQEntryPtr entry,
ExtendedServerEventRecord* event);
```

To make your code work on a PowerPC, you need to create a `ServerEventHandlerUPP` by calling `NewServerEventHandlerProc`:

```
ServerEventHandlerUPP NewServerEventHandlerProc
(ServerEventHandlerProcPtr yourProc);
```

When you're done, call `DisposeRoutineDescriptor`:

```
void DisposeRoutineDescriptor (ServerEventHandlerUPP yourUPP);
```

Index

M, N, O

P

Q

R

S, T

U

V

W–Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Kathy Wallace

ILLUSTRATOR
Dave Arrigoni

PRODUCTION EDITOR
Gerri Gray

Special thanks to Erik Sea