NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note NW515
AppleShare Q&As

This Technical Note contains a collection of archived Q&As relating to a specific topic - questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Oct 01 1990]

Alternate User Authentication Methods (UAMs)

Date Written: 3/9/93

Last reviewed: 6/24/93

Is there a sanctioned way to use `PBVolumeMount` to mount a volume with an alternate user authentication method (UAM)? I heard there's an old document describing AppleShare UAMs.

Alternate User Authentication Methods (UAMs) aren't a supported feature of the AppleShare workstation software. The only documentation to ever come out of Apple concerning alternate UAMs is a draft proposal titled "Multiple User Authentication Methods for AppleShare" and it was written back in 1987. The thing to remember is that the document was a *draft* proposal; not final documentation for a *tested* mechanism in shipping product. Apple Developer Technical Support will *not* supply this document to developers who don't have it.

Although the code for the alternate UAM mechanism described by that document is in the AppleShare workstation, it has never been fully implemented or tested in any version of AppleShare Workstation (including the AppleShare 3.0 Workstation); it has been only partially tested by some of the engineers that wrote that code. Because the code for alternate UAMs has never been officially tested or documented as a shipping product, Apple Developer Technical Support can't officially support you in this area. On top of that, engineering doesn't consider the alternate UAM mechanism part of any current project, so we (in DTS) have no project to submit bugs against. It is *very* likely that the code for the alternate UAM mechanism in the current AppleShare workstation will be removed at some point, so you should not rely on it in any of your programs.

Since alternate UAMs aren't a supported feature of AppleShare workstation, new features added to the AppleShare workstation don't support them, and other parts of the operating system don't use them. So `PBVolumeMount` (which is implemented for AppleShare volumes by the AppleShare workstation's external file system code) doesn't support alternate UAMs, and the Alias Manager (which uses `PBVolumeMount` to mount AppleShare volumes) doesn't know anything about alternate UAMs and has no way to support them.

Determining AppleShare Admin password

Date Written: 2/8/93

Last reviewed: 7/9/93

I mistyped an AppleShare file server's administrative password. How can I retrieve it?

We can't provide the information you've requested since it is considered proprietary (for security reasons). If you've lost your admin password you'll need to delete your current Users & Groups Data File. This is the only sanctioned way to reset your password. (If you have a duplicate Users & Groups file on another server, you can copy this file over and change the password accordingly.)

CreateResFile calls and AppleShare drop folders

Date Written: 4/28/93

Last reviewed: 7/2/93

Why does calling `FSpCreateResFile` return an `afpAccessDenied` error (in `ResErr`) when the destination folder is an AppleShare drop folder?

Using any of the `CreateResFile` calls in a drop box is a useless exercise. Here's why: The access privileges within a drop box are very limited write-only access. This lets you create new files and then perform a small set of operations on the file while it's empty (no bytes in either the data or the resource fork of the file). So, while the file is empty, you can open it (either fork) with write-only access (`PBHOpenDeny` or `PBHOpenRFDeny`), and write the file's attributes (`PBHSetFInfo` or `PBSetCatInfo`), including Desktop Manager comments. Once either fork has a single byte of data written to it, you can no longer open or change the file's attributes. This makes it possible to copy a file into a drop folder, but not to manipulate or delete a file (containing data) that's already in the drop folder. You can also move a file or directory into a drop folder if that file or directory is already on the same server volume.

So, when you call `CreateResFile` on a drop folder, a new file is created and data is added to the resource fork of the file. Since the file now has data in one of the forks, you cannot open the file or change any of the file's attributes. `FSpCreateResFile` fails because after performing the `CreateResFile` operation that writes data into the resource fork, it attempts to set the file's attributes (creator, fileType, and scriptTag).

For a quick explanation of drop folder access privileges and rules, see the Macintosh Technical Note "Creating Files Inside an AppleShare Drop Folder" (Files 18). For a complete explanation of what AFP access privileges you need to perform specific operations on an AppleShare (AFP) server, see the section "Directory Access Control" in *Inside AppleTalk* starting on page 13-31.

Updating aliases when File Sharing state changes

Date Written: 2/4/93

Last reviewed: 7/2/93

We create an alias for a file when File Sharing is off, so the alias doesn't contain server and zone information. Then we turn on File Sharing and resolve the alias. The file, of course, is found but the alias isn't marked for an update. Shouldn't it be updated?

According to *Inside Macintosh: Files,* `wasChanged` is set to TRUE only on aliases created by `NewAlias` (not `NewAliasMinimalFromFullPath` or `NewAliasMinimal`) when it sees that key information has changed. The key information is:

- name of the target
- directory ID of the target's parent
- file ID or directory ID of the target
- name and creation date of the volume on which the target resides

Since none of that changes when you turn File Sharing on or off, the `wasChanged` flag isn't set to TRUE. If you're really worried about it, just call UpdateAlias every time you use the alias (unless you think this would be a major performance hit). Or maybe you should update it only if you notice that it doesn't have server information and the server is now turned on (to check for this, call `PBHGetVolParms` and check the `bHasPersonalAccessPrivileges` bit).

Macintosh AppleShare client version details

Date Written: 12/18/92

Last reviewed: 3/1/93

System 7.0.1 doesn't seem to correctly display (or display at all) login (greeting) messages from an AppleShare server. System 7.1 works fine. Is this a known problem, and is there anything we can do do get the login messages to show up?

Not all Macintosh AppleShare clients support server messages (including the greeting messages). Every version of the Macintosh AppleShare client released since AppleShare 3.0 has supported server messages. If you install the AppleShare client version 3.0, 7.1 or 3.0.1 on System 7.0, 7.0.1 with or without System 7 Tune-up, you'll get server greeting messages.

Here's a list of Macintosh AppleShare client versions in the order they were released (notice that they aren't in version order). We've listed the important changes made to each version and listed system software compatibility. (As you might suspect, DTS doesn't have a complete list of all minor changes made to every version.)

```
AppleShare Client version:  2.0
Shipped with:               System 6.0.x and AppleShare 2.0
System Software versions:   6.0.x
Maximum AFP version:        2.0
Features:                   Supports AFP 2.0

AppleShare Client version:  7.0
Shipped with:               System 7.0
System Software versions:   7.0
Maximum AFP version:        2.1
Features:                   Supports required AFP 2.1 features only. Adds
                            support for these new File Manager functions:
                               PBCreateFileIDRef
                               PBDeleteFileIDRef
                               PBResolveFileIDRef
                               PBExchangeFiles
                               PBCatSearch
                               PBGetVolMountInfoSize
                               PBGetVolMountInfo
                               PBVolumeMount (no volume password support)

AppleShare Client version:  7.0.1
Shipped with:               System 7.0.1
System Software versions:   7.0 or 7.0.1
Maximum AFP version:        2.1
Features:                   No changes from version 7.0

AppleShare Client version:  3.0
Shipped with:               AppleShare 3.0
System Software versions:   System 6.0.4 through 6.0.8, System 7.0 and 7.0.1
Maximum AFP version:        2.1
Features:                   Added support for server greeting messages.
                            Patched the System 6 File Manager to support
                            these File Manager functions:
                               PBGetVolMountInfoSize
                               PBGetVolMountInfo
                               PBVolumeMount
                            PBVolumeMount now supports volume passwords.

AppleShare Client version:  7.1 and 3.0.1
Shipped with:               System 7.1 and AppleShare 3.0.1
System Software versions:   System 6.0.4 through 6.0.8, System 7.0, 7.0.1,
                            and 7.1
Maximum AFP version:        2.1
```

Auto-mounting AppleShare 3.0 volumes when a Greeting is enabled

Date Written: 6/5/92

Last reviewed: 3/1/93

Our program calls PBVolumeMount using MPW 3.2.2 interfaces to automount FileShare and AppleShare servers when performing unattended backups, but if an AppleShare 3.0 server has a Greeting, the backup cannot proceed until the Greeting is dealt with. Is there a way to auto-mount these volumes which suppresses the Greeting?

To mount a volume with `PBVolumeMount` and disable the greetings, you need to set bit 0 in the flags field of the `AFPVolMountInfo` record. The flags field was marked reserved in *Inside Macintosh* Volume VI because the greeting message feature had yet to be released when Volume VI was written. The Inside Macintosh: Files book (the revised edition of Inside Macintosh) documents this flag bit.

One thing you should be aware of is an alias record to an AFP volume created by the Alias Manager stores the flags information. That is, if you mount a volume with greetings disabled, create an alias to the volume (or a file or directory on the volume), and then later mount the volume by resolving the alias, the greeting message will be disabled. If the volume is mounted with the flags word cleared, any aliases resolved with show greeting messages (if any).

Here's a short sample application that mounts a volume with greeting messages disabled:

```
Program MountVolTest;

USES
  Types,
  Files;

CONST
  kNormalMountFlags = 0;
  kInhibitMsgFlags = 1;

TYPE
  Str8 = STRING[8];

  { I like the following record structure better than }
  { the AFPVolMountInfo struture in Files.p }
  MyAFPVolMountInfo = RECORD
    length: Integer; { length of this record }
    media: VolumeType; { type of media, always AppleShareMediaType }
    flags: Integer; { 0 = normal mount; 1 = no greeting messages }
    nbpInterval: SignedByte; { NBP interval ; 7 is a good choice }
    nbpCount: SignedByte; { NBP count ; 5 is a good choice }
    uamType: Integer; { 1 = 'No User Authent' (guest); }
                      { 2 = 'Cleartxt Passwrd'; }
                      { 3 = 'Randnum Exchange'; }
                      { 6 = '2-Way Randnum exchange' }
    zoneNameOffset: Integer; { offset-record start to zoneName }
    serverNameOffset: Integer; { offset-record start to serverName }
    volNameOffset: Integer; { offset-record start to volName }
    userNameOffset: Integer; { offset-record start to userName }
    userPasswordOffset: Integer; { offset-record start to userPassWord }
    volPasswordOffset: Integer; { offset-record start to volPassWord }
    zoneName: Str31; { server's AppleTalk zone name }
    serverName: Str31; { server name }
    volName: Str27; { volume name }
    userName: Str31; { user name (0 length = guest) }
    userPassWord: Str8; { user password (0 length = no user password) }
    volPassWord: Str8; { volume password (0 length = no volume password) }
    END;

VAR
  gUAMType: Integer;
  gZoneName: Str31;
  gServerName: Str31;
  gVolName: Str27;
  gUserName: Str31;
  gUserPassWord: Str8;
  gVolPassWord: Str8;

  FUNCTION SilentMountAFPVolume: OSErr;
    VAR
      theAFPInfo: MyAFPVolMountInfo;
      pb: HParamBlockRec;
  BEGIN
    WITH theAFPInfo DO
      BEGIN
        length := sizeof(MyAFPVolMountInfo);
        media := AppleShareMediaType;
        flags := kInhibitMsgFlags;
        nbpInterval := 7;
        nbpCount := 5;
        uamType := gUAMType;
        zoneNameOffset := ORD4(@zoneName) - ORD4(@theAFPInfo);
        zoneName := gZoneName;
        serverNameOffset := ORD4(@serverName) - ORD4(@theAFPInfo);
        serverName := gServerName;
        volNameOffset := ORD4(@volName) - ORD4(@theAFPInfo);
        volName := gVolName;
        userNameOffset := ORD4(@userName) - ORD4(@theAFPInfo);
        userName := gUserName;
        userPasswordOffset := ORD4(@userPassWord) - ORD4(@theAFPInfo);
        userPassWord := gUserPassWord;
        volPasswordOffset := ORD4(@volPassWord) - ORD4(@theAFPInfo);
        volPassWord := gVolPassWord;
      END;
    pb.ioBuffer := @theAFPInfo;
    SilentMountAFPVolume := PBVolumeMount(@pb);
  END;

BEGIN
  gUAMType := 1; { mount as guest }
  gZoneName := 'Dev Support Center (DTS)';
  gServerName := 'Briggs';
  gVolName := 'SevenOh';
  gUserName := ''; { guest }
  gUserPassWord := ''; { guest }
  gVolPassWord := ''; { no volume password }
  IF SilentMountAFPVolume <> noErr THEN
    DebugStr('couldn''t mount volume');
```

Mounting volumes without using aliases

Date Written: 9/25/92

Last reviewed: 11/24/92

How can I mount a volume without using aliases? I get the mounting information, then attempt to mount the volume. However, the `PBVolumeMount` call returns an error code.

The `PBGetVolMountInfo`, `PBGetVolMountInfoSize`, and `PBVolumeMount` functions are currently handled by only the AppleShare external file system (part of the AppleShare Chooser extension). Those functions are available on AppleShare volumes when the AppleShare Chooser extension is version 7.0 (system software versions 7.0 and 7.0.1), version 3.0 (AppleShare 3.0), or version 7.1 (System 7.1). The AppleShare Chooser extension version 3.0 can be installed on System 6 systems, and then the `PBGetVolMountInfo`, `PBGetVolMountInfoSize`, and `PBVolumeMount` functions can be used in System 6. Other file systems may support these functions in the future. The paramErr error code is returned when these functions aren't available on a particular volume.

How to tell if someone else has your data file open

Date Written: 7/7/92

Last reviewed: 11/1/92

How can I tell if another person has my data file open? According to *Inside Macintosh* Volume IV (pages 148-149), `PBGetFInfo` will tell you if anyone has a file open by returning the first access path found in `ioFRefNum`, but if I have this file already open, it returns information about myself. I want to be able to tell if anyone *else* has it open.

___

If the file is on a local volume, you can index through the open FCBs to find all open connections to the file. You'll find your own connection in that list but since you know what your `ioRefNum` is, you can ignore that match. For example, the following function will tell you if someone else has the file specified by myRefNum open. With a small change, you could have it return the number of other open connections instead of TRUE or FALSE.

```
FUNCTION OthersHaveItOpen (myRefNum: Integer): Boolean;
   VAR
      err: OSErr;
      fcbPB: FCBPBRec;
      myName: Str255;
      myVRefNum: Integer;
      myDirID: LongInt;
      index: LongInt;
      fileName: Str255;
      found: Boolean;
BEGIN
   fcbPB.ioNamePtr := @myName;      { let PBGetFCBInfo fill in the name. }
   fcbPB.ioVRefNum := 0;            { look on all volumes. }
   fcbPB.ioRefNum := myRefNum;      { use myRefNum to get info. }
   fcbPB.ioFCBIndx := 0;            { use ioRefNum instead of ioFCBIndx. }
   err := PBGetFCBInfoSync(@fcbPB); { get the open file's info. }
   IF err = noErr THEN
      BEGIN
         { save my file's vRefNum and dirID for matching later. }
         { myName was filled in by PBGetFCBInfo call above. }
         myVRefNum := fcbPB.ioFCBVRefNum;
         myDirID := fcbPB.ioFCBParID;

         { index through the open files on the volume }
         index := 1;
         REPEAT
            fcbPB.ioNamePtr := @fileName;
            fcbPB.ioVRefNum := myVRefNum;
            fcbPB.ioRefNum := 0;
            fcbPB.ioFCBIndx := index;
            err := PBGetFCBInfoSync(@fcbPB);
            IF err = noErr THEN
               { see if there is a match that isn't myRefNum }
               found := (fileName = myName) AND
                        (fcbPB.ioFCBParID = myDirID) AND
                        (fcbPB.ioRefNum <> myRefNum)
            ELSE
               found := FALSE;  { no matches on errors }
            index := index + 1; { next index position... }
         UNTIL found OR (err <> noErr);

         OthersHaveItOpen := found;
      END
   ELSE
      OthersHaveItOpen := FALSE; { we don't even have it open! }
```

On nonlocal volumes (AppleShare volumes), you can use PBHOpenDeny with the deny-read and deny-write permissions to ensure nobody else can open a file you have open. You can tell which volumes are nonlocal by calling PBHGetVolParms and looking at the value returned in the GetVolParmsInfo.vMServerAdr field. If that value is 0, the volume is local.

Unmounting volumes shared with Macintosh File Sharing

Date Written: 9/15/92

Last reviewed: 3/10/93

I tried to unmount a volume shared with Macintosh File Sharing from my program using the following steps: I shut down the file service with the SCShutDown server control call; I call SCPollServer to make sure the file service is really off (scServerState = SCPSJustDisabled); then, I call PBUnmountVol to attempt to unmount the volume. It didn't work because PBUnmountVol fails with fBsyErr (-47). I broke on the _UnmountVol trap because the AppleShare PDS file, where the file server keeps the access privilege and share-point information for the shared volume, was open. Why is AppleShare PDS still open when I've turned the file service off? How can I close it and unmount the volume?

SCPollServer returns the state of the file service, not the file server application (in this case, File Sharing Extension is the file server application). When SCPollServer returns a server state of SCPSJustDisabled, the file service is off; however, the file server application may or may not still be running. The AppleShare PDS file will eventually get closed before the file server application quits.

There's an easy way to determine when the File Sharing application has quit (and thus when the AppleShare PDS file is closed): just use the Process Manager GetNextProcess and GetProcessInformation calls to find out when File Sharing Extension is no longer running. The File Sharing Extension application has a processType of 'INIT' and a processSignature of 'hhgg'. Here's a function you can use to see if the File Sharing Extension application is running:

```
FUNCTION FileSharingAppIsRunning: Boolean;
  CONST
    FileSharingSignature = 'hhgg';  {Macintosh File Sharing}
  VAR
    err:      OSErr;
    myPSN:    ProcessSerialNumber;
    myPInfoRec:  ProcessInfoRec;
BEGIN
  myPSN.highLongOfPSN := 0;  {Start at beginning of process list}
  myPSN.lowLongOfPSN := kNoProcess;
  myPInfoRec.processInfoLength := sizeOf(ProcessInfoRec);
  myPInfoRec.processName := NIL;   {Don't need process name}
  myPInfoRec.processAppSpec := NIL; {Don't need process location}
  FileSharingAppIsRunning := FALSE; {Haven't found it yet}
  WHILE (GetNextProcess(myPSN) = noErr) DO
    IF GetProcessInformation(myPSN, myPInfoRec) = noErr THEN
      IF (myPInfoRec.processSignature = FileSharingSignature)
        THEN
        FileSharingAppIsRunning := TRUE;  {Found it}
```

After shutting down the file service, your event loop will need to poll with FileSharingAppIsRunning because you must give the file server application processing time to close files, dispose of memory, and perform other shutdown operations. If you poll with FileSharingAppIsRunning without giving other processes time, File Sharing will never shut down.

Maximum volumes for file sharing

Date Written: 3/9/92

Last reviewed: 8/1/92

In the past I've been able to file share more volumes off my Macintosh SCSI storage devices than I can with System 7. Now I get an alert saying: "One or more items could not be shared because not all volumes are available for file sharing." Please advise as to what the problem might be.

Macintosh File Sharing will only prepare for sharing the first 10 volumes it sees (it enumerates the volume list with PBHGetVInfo). The volumes you can't share will usually be the ones mounted last. The reason you used to be able to share another set of volumes probably has to do with some change you've made (like changing the boot volume or a volume's SCSI ID number).

So, you've just hit the limits of File Sharing. The solution to your problem is to use AppleShare 3.0 -it will share up to 50 volumes. File Sharing wasn't intended to be the end-all in file servers; it was designed for individuals who want to occasionally share files with a small number of other users. Here are some limits to File Sharing that you should note:

- The number of users and groups in the Users & Groups Data File is limited to 100 total. (The limit with AppleShare 3.0 is 8192 total.)
- The number of users that can be logged in at one time is 10 (this doesn't count the owner of the system, one remote connection is always reserved for the owner of the system). (The limit with AppleShare 3.0 is 120.)
- The number of share points available for regular users is 10. (The limit with AppleShare 3.0 is 50.)

- The number of sharable volumes (what the owner sees when he or she logs in remotely and what can be shared or partially shared) is 10. (The limit with AppleShare 3.0 is 50.)

AppleShare 3.0 also supports many other user (for example, server messages), security, and developer features (server control calls and the server event mechanism) not supported by Macintosh File Sharing.

AppleShare user limit

Date Written: 11/16/90

Last reviewed: 12/19/90

What is the maximum number of users that can be logged in to any one AppleShare file server? What can we do to increase the limit? Will upgrading to AppleTalk Phase II help? Is there an upgrade to the AppleShare 2.0.1 software?

AppleShare currently has a limit of 50 simultaneous users. This is a limitation in the software and is not related to AppleTalk. Changing from AppleTalk Phase I to Phase II will not change anything. The next version of AppleShare might raise this limitation. We do not have any projected dates for a release of the next version of AppleShare. You may want to periodically check with APDA for any update or new release.

Maximum number of users supported by AppleShare for each CPU

Date Written: 5/3/89

Last reviewed: 11/21/90

What is the maximum number of users supported by AppleShare? Does this number change based on the type of CPU being used for the server?

The following chart lists some current AppleShare limits (AppleShare 1.1, 2.0, and 2.01) which are based upon the chosen server platform and memory configuration. The limits that otherwise might be present on a workstation are still in effect, and are not affected by the workstation being logged in to an AppleShare server. These limits will change in the future.

Server machine is Macintosh Plus, SE, or II with 1 MB:

- Number of users: 25
- Number of locked ranges: 1000
- Number of open files: 80
- Number of volumes: 16

Server machine is Macintosh II with more than 1 MB:

- Number of users: 50
- Number of locked ranges: 2000
- Number of open files: 160
- Number of volumes: 16

X-Ref:

Macintosh Technical Note "AppleShare 1.1 and 2.0 Limits"

Macintosh file system active ranges

Date Written: 3/18/91

Last reviewed: 6/7/91

How many active ranges can a Macintosh application have on a shared file? If the answer is more than one, is the limit per application or per machine? If two ranges overlap, are they joined into one range? Can an application nest ranges? For example, if an application's user performs an action that forces a record to be locked and later the application locks the full range of the file, does the initial record lock disappear?

The only way to determine the limit is to hit the limit and get a NoMoreLocks error. The number of range locks supported is a limit of the server platform, and that limit is shared by all users of the server (at least it is with Apple's AppleShare server software). With Apple's server-based version of AppleShare, approximately 40 locks per user are allowed (for example, if the server allows 25 users, there are 1000 locked ranges total; if the server allows 50 users, there are 2000 locked ranges total; and with File Sharing running under System 7.0, approximately 20 locks are allowed per user). Other vendors may allow more or fewer locked ranges on their implementations of an AppleTalk Filing Protocol (AFP) server. Notice that the numbers given are per user, not per application. It's assumed that a user probably won't need more than a few locks at a time on a single file.

You cannot have range locks that overlap. You'll get a `RangeOverlap` error from AFP. All the rules for range locking can be found in the AFP chapter of *Inside AppleTalk* (page 13-56). Additional information on AppleShare limits is available on the latest *Developer CD Series* disc.

The Macintosh Technical Note "Lock, Unlock the Range" covers several important details about `PBLockRange` and `PBUnlockRange` that are not in *Inside Macintosh.*

AppleShare open file limit

Date Written: 10/8/91

Last reviewed: 10/8/91

On an AppleShare 2.0 File Server platform, the only application that can access files outside of the Server Folder (that is, the System Folder) is the file server application. AppleShare Foreground applications (described in the Macintosh Technote "AppleShare Foreground Applications" are the only other applications that should be running on a server and they can only access files inside the Server Folder. All file forks (referred to as files from here on) opened by remote AppleShare workstations are opened by the File Server application.

The File Server application will open a file only one time. All access to that file from any number of workstations will use the single access path the File Server has opened. Only when all workstations have closed the file does the File Server really close the file on the server. So, that means only one FCB is used on the server per open file, even if 50 users have shared access to that file.

The File Server application handles all access control to an open file using the AppleTalk Filing Protocol (AFP) deny-mode permission model. The only reason a user won't be able to open an access path to a file on a server is if another user has opened that file with a deny-mode that conflicts with the second user's request, or the user does not have the access rights needed to open files in the file's parent directory or directory ancestors. The AFP deny-mode permission model is described briefly in *Inside Macintosh* Volume V, File Manager Extensions in a Shared Environment, and in detail in the AppleTalk Filing Protocol chapter of *Inside AppleTalk.*

As noted in the Macintosh Technote "AppleShare 1.1 and 2.0 Limits," the maximum number of open file forks on an AppleShare 2.0 server is either 80 on a 1 MB MC68000 server platform, or 160 on a server platform with more than 1 MB and a MC68020 or greater processor. That figure includes the number of files kept open by the system and the file server application. If an AppleShare Foreground application is running on the server (for example, the AppleShare Print Server), then any files it may have open count against the maximum, too. The same can be said for open desk accessories. This Technote currently doesn't say anything about those files counting towards the limit. If the 160 (or 80) file limit is a problem, you can use the "Up Your FCBs" INIT to bump the number available up to the maximum (342) allowed by the File Manager. "Up Your FCBs" can be found on AppleLink in the Developer Support: Developer Technical Support: Hacks folder.

Software-selecting an AppleShare volume

Date Written: 10/23/90

Last reviewed: 2/20/91

Is there any source code available for mounting/unmounting AppleShare volumes?

There are actually a couple of ways to select an AppleShare volume. You could use the Choose tool in MPW that accomplishes this, or you can do it with aliases under System 7. Other than the MPW tool, there is no other supported way of doing this under pre-7.0 systems; there are no current hooks to allow easy mounting of AppleShare volumes programmatically. It gets pretty nasty trying to figure out everything that is necessary to accomplish this, which is why people here pretty much stay away from this as well. Also, some low-level stuff may be proprietary, which is why the tool is supplied for developers. The Choose tool is described in the MPW docs. It should be pretty straightforward to use.

2.0.1 PBHGetDirAccess and PutDirAccess restrictions

Date Written: 12/5/90

Last reviewed: 1/16/91

If a volume is connected to an AppleShare server, but is not an AppleShare volume, will the `PBHGetDirAccess` (and `PutDirAccess`) function work on it? Can an INIT on the server make these calls?

For AppleShare 2.0.1, the INIT cannot make these calls on non-file server volumes. In future versions, the `PBHGetDirAccess` and `PutDirAccess` calls can safely be made on all volumes connected.

Server Move & Rename folder

Date Written: 12/5/90

Last reviewed: 6/14/93

A folder is created in the Server Folder called "...Move & Rename." What is this and what are its contents? Should it be backed up? Are there any other temporary folders and files that might need to be backed up?

It's the "$01$02$03Move & Rename" folder that AppleShare 2.0.1 and future versions create for the two-step process of moving and renaming a file or a folder, a feature that is not provided via HFS. It should be backed up, but in general will not contain anything. (It has something in it only for a brief instant and only if the server has IBM PCs or some other computer that uses this call. Macintosh systems don't.) It needs to be backed up for 2.0.1 so that the folder is there if the server is restored. (Otherwise, Admin will have to be run to create a new one, a somewhat disconcerting action to perform after completely restoring a file server.)

How to tell if application's running on a server

Date Written: 12/5/90

Last reviewed: 1/16/91

What is the best way to determine if a Macintosh application is running on a server?

For 2.0.1 you can test the longword at $B50. If it is 0 or -1, the server is not running. If it isn't--that is, it's a real address--then the server is either starting up, or is running. There might be a hook available in future versions of AppleShare that your process can hook into.

Detecting AppleTalk being closed down by user

Date Written: 12/12/90

Last reviewed: 6/14/93

How do I detect that a user has closed down my AppleTalk connection (by turning AppleTalk off from the Chooser or by changing network connections from the Network control panel)?

The AppleTalk Transition Queue provides a means to determine when the AppleTalk drivers change status or when they might be closed in the very near future. The Transition Queue informs its clients (everyone who has asked to be added to the queue) each time the state of the .MPP driver changes state (opened or closed) or is about to change state.

The AppleTalk Transition Queue is documented in *Inside Macintosh* Volume VI, Chapter 32 (The AppleTalk Manager) and is also documented in the *Macintosh AppleTalk Connections Programmer's Guide,* Chapter 3 (Calls to the LAP Manager), available from APDA. A sample AppleTalk Transition Queue routine is shown in Technote 311.

Purpose of AppleShare SP file

Date Written: 3/14/91

Last reviewed: 4/29/91

What is the file "AppleShare SP" and what does it do? The AppleShare File Server seems to run without it, and it reduces the alert sound to just a beep even when the server is not running on that computer. Is the file really needed?

You do need the AppleShare SP INIT on your file server. The AppleShare SP (Small Patch) INIT is designed to correct a minor incompatibility between AppleShare 2.0x and the Sound Manager. The INIT forces the Macintosh to use the "Simple Beep" sound at all times. System 7.0 file sharing and future versions of AppleShare do not need the INIT.

Macintosh EOF in an AppleShare environment

Date Written: 3/18/91

Last reviewed: 6/10/91

I ran into the following when updating the logical end of file (EOF) of a shared file: Application A and Application B have access to a file under AppleShare. Each is using `fsRdWrShPerm`. When Application A changes the logical EOF, Application B doesn't seem to notice that EOF has changed until Application B calls `GetEOF`. Is there a better way to make Application B aware of the change of logical EOF?

You've made a correct assumption that the correct way to keep track of EOF in an AppleShare environment is to ask for it. When you open a file, the AppleShare workstation code translates the Macintosh operating system Open call to the AFP (AppleTalk Filing Protocol) `FPOpenFork` call, and sets the bits in the bitmap parameter to current length of the fork opened (in the case of Open, the data fork). It then uses THAT as the EOF for future operations unless it gets an update from the server. Because the server does not constantly update everyone who has the file open, you have to ask to find if another user (or application) has made a change. Just remember that using `GetEOF` will only get you the EOF at that instant in time. Someone else sharing the file could change EOF right after you check it.

The `PBLockRange` function can be used by an AppleShare aware application to prevent another user from appending data to a shared file while you are appending data. For example:

```
paramBlock.ioRefNum := myFileRef;
err := PBGetEOF(@paramBlock, FALSE); {get the current EOF}
{check for errors in a real application}

oldEOF := paramBlock.ioMisc; {save the current EOF}
paramBlock.ioReqCount := -1; {$FFFFFFFF}
paramBlock.ioPosMode := fsFromStart;
paramBlock.ioPosOffset := oldEOF; {start range lock at current EOF}
err := PCLockRange(@paramBlock, FALSE); {and lock the rest of the fork}
{check for errors in a real application}

{now you can append data to the file}

paramBlock.ioRefNum := myFileRef;
paramBlock.ioReqCount := -1; {$FFFFFFFF}
paramBlock.ioPosMode := fsFromStart;
paramBlock.ioPosOffset := oldEOF;
err := PCUnlockRange(@paramBlock, FALSE); {unlock the locked range}
```

`PBLockRange` can also be used when you need to truncate a shared file. Locking the portion of the file you're about to truncate prevents another user from using that portion during the truncation process. For example:

```
  paramBlock.ioRefNum := myFileRef;
  paramBlock.ioReqCount := -1; {$FFFFFFFF}
  paramBlock.ioPosMode := fsFromStart;
  paramBlock.ioPosOffset := theNewEOF; {start lock at truncation point}
  err := PCLockRange(@paramBlock, FALSE); {and lock the rest of the fork}
  {check for errors in a real application}

  paramBlock.ioMisc:= theNewEOF;
  err := PBSetEOF(@paramBlock, FALSE); {set the new EOF (truncate the file)}
  {check for errors in a real application}

  paramBlock.ioReqCount := -1; {$FFFFFFFF}
  paramBlock.ioPosMode := fsFromStart;
  paramBlock.ioPosOffset := theNewEOF;
  err := PCUnlockRange(@paramBlock, FALSE); {unlock the locked range}
  {check for errors in a real application}

The entire fork can be locked with:

  paramBlock.ioRefNum := myFileRef;
  paramBlock.ioReqCount := -1; {$FFFFFFFF}
  paramBlock.ioPosMode := fsFromStart;
  paramBlock.ioPosOffset := 0; {lock from the beginning}
  err := PCLockRange(@paramBlock, FALSE); {lock the whole fork}
  {check for errors in a real application}

  {do your thing}

  paramBlock.ioRefNum := myFileRef;
  paramBlock.ioReqCount := -1; {$FFFFFFFF}
  paramBlock.ioPosMode := fsFromStart;
  paramBlock.ioPosOffset := 0; {unlock from the beginning}
  err := PCUnlockRange(@paramBlock, FALSE); {unlock the whole fork}
```

Macintosh AppleShare versus file sharing capabilities

Date Written: 4/3/91

Last reviewed: 6/21/91

We are using the file sharing capabilities of Macintosh systems with System 7.0 to make them mini file servers. Where can information that details the features of both AppleShare and file sharing be found? We are considering using only file sharing in the office if it is capable of providing most of what AppleShare provides.

Information on file sharing can be found in the System 7 Personal Upgrade Kit and in *Inside Macintosh* Volume VI, on your *Developer CD Series* disc. The maximum number of concurrent connections allowed on a Macintosh using file sharing is 10. The performance of an AppleShare file server (the standard kind of server) is approximately 25 percent better than a similar configuration of Macintosh computers acting as a file sharing server.

System 7 and AppleTalk Internet Router

Date Written: 9/17/91

Last reviewed: 11/25/91

We've tried to run the Apple Internet Router with our System 7 file sharing servers. There does not appear to be support for multiple networks. Is there some solution to this?

The AppleTalk Internet Router and System 7 are compatible, with two exceptions: virtual memory and 32-bit addressing. You need to drag-install it instead of using the Installer. The Installer script on the router disk will put the parts of the router in the wrong place. Here are the steps to drag-install the router:

1. Drag the files Router, LocalTalk (Built-in), and LocalTalk (Modem) from the System Folder on the *AppleTalk*

Internet Router disk to the closed System Folder on the disk where you want to install. System 7 will automatically put the Router file in the System Folder, and put the LocalTalk (Built-in) and LocalTalk (Modem) files in the Extensions folder in the System Folder.
2. Under System 7, open the System file on the *AppleTalk Internet Router* disk (double-click the System file). A window will open showing the desk accessories, fonts, and sounds that are part of that System file. Drag the Router desk accessory from that window to the closed System Folder on the disk where you want to install. System 7 will automatically put the Router desk accessory in the Apple Menu Items folder in the System folder.
3. Reboot.

That's all there is to it.

X-Ref:

"System 7: Installing Internet Router 2.0," AppleLink Tech Info Library

TMGetTermEnvirons envVersTooBig error

Date Written: 9/17/91

Last reviewed: 9/17/91

A call made to `TMGetTermEnvirons` returns -5502 or `envVersTooBig`. The call is made with a good terminal handle started from your TTY tool. What's wrong? How do I fix it?

When you call `TMGetTermEnvirons`, the `TermEnvironRec` that you hand it a pointer to must have initialized the version field with `curTermEnvRecVers`. The following will work properly:

```
void getTEnvirons(TermHandle aTerm)
{  ...
   TermEnvironRec tEnv;
   TMErr err;
   ...
   tEnv.version = curTermEnvRecVers;
   err = TMGetTermEnvirons(aTerm,&tEnv);
   /* check for errors, do whatever... */
   ...
```

This is consistent with most other Macintosh "get environment" calls such as `SysEnvirons`, and is documented (somewhat unclearly) at the top of page 113 in *Inside the Macintosh Communications Toolbox.*

Modifying a server volume's backup date-time from a workstation

Date Written: 12/5/91

Last reviewed: 1/27/92

How can I change the backup date of a remote AppleShare volume? When I get the volume information with `PBHGetVInfo` (followed by a `PBFlushVol`), change the backup date field, and call `PBSetVInfo`, the date is changed in my local copy of the volume information, but when I unmount and remount the AppleShare volume, the original backup date is still there.

On an AppleTalk Filing Protocol (AFP) file server, two of the volume date-time values, the volume creation date-time and the volume modification date-time, are managed solely by the server and can't be changed by workstations. The third volume date-time value, the volume backup date-time, can be set by a workstation with only one AFP call, `afpSetVolParms`. However, the File Manager, through the Macintosh AppleShare external file system, does not give an application a way to make the `afpSetVolParms` call. That leaves only one way you can change a server volume's backup date-time from a workstation: You'll have to use the AppleTalk .XPP driver to access the server directly.

Using the .XPP driver to change the backup date-time involves these steps:

- Open the .XPP driver and get the driver reference number.
- Use the `afpLogin` variant of AFPCommand to start a session and log in to the server. If the "Randnum Exchange" or "2-Way Randnum Exchange" user authentication methods are used, you will receive an `AuthContinue` error (-5001) from the `afpLogin` call and you'll have to follow up the `afpLogin` call with an `afpLoginCont` call (through `AFPCommand` again) to finish the log-in sequence.
- Once you're logged in to the server, you need make an `afpGetSrvrParms` call to get a list of volumes and to find

out if the volume you're interested in has a password associated with it.
- Then you need to call `afpOpenVol` with the volume name (and password if there is one). You can have `afpOpenVol` give you the volume's current backup date-time and other volume information if you set the appropriate bits in the bitmap parameter passed to `afpOpenVol`.
- Now that you're logged in to the server and have the volume opened, you can make an `afpSetVolParms` call to change the backup date-time.
- After changing backup date-time, you need to close the volume with `afpCloseVol`, and then log off the server with `afpLogout`.

The .XPP driver's AFP commands are described in *Inside Macintosh* Volume V in the AppleTalk chapter (pages V-524 through V-550). For a description of the AFP calls, user authentication methods, and other AFP information, you need to look in the AppleTalk Filing Protocol chapter of *Inside AppleTalk.* If you decide you want to really use the .XPP driver as described above and want to use the 2-Way Randnum Exchange user authentication method supported by System 7 File Sharing and AppleShare 3.0, contact DTS for a preliminary version of the AFP 2.1 specification that describes that new authentication method.

AppleShare Print Server 3.0 and AppleTalk self-send

Date Written: 2/28/92

Last reviewed: 2/28/92

When I issue a `PAPWrite` from my application to the AppleShare Print Server 3.0 running on the same Macintosh, `PAPWrite` locks up in a tight loop. If I send to a LaserWriter or the AppleShare Print Server 3.0 running on a different Macintosh system, all works well. The LaserWriter Font Utility 7.0 behaves the same as my application: It works if the spooler is remote and locks up if the spooler is local.

You're probably calling `PAPWrite` and then not giving up any system time needed by the print server to process the data you sent to it. That just doesn't work in the self-send environment. For example, the following won't work:

```
    PAPWrite(refNum, writeBuff, dataSize, eof, compState);
    WHILE compState = 1 DO
```

What your application should do is drop back into its event loop after making the `PAPWrite` call and then poll `compState` to see when the `PAPWrite` completes. By calling `WaitNextEvent` from your event loop, your application gives the print server application the time it needs to receive and process the data you sent to it.

The LaserWriter Font Utility wasn't designed to work with print servers and will exhibit the same problem your application is experiencing.

System 6 & 7 Chooser AppleShare differences

Date Written: 2/28/92

Last reviewed: 2/1/93

What's the limit on the number of servers displayable in the Chooser as well as the maximum number of AppleShare mount points per server? Are the System 7 limits the same as for System 6?

Most System 6 Chooser limitations have been eliminated with the System 7 Chooser. Here's how each version operates:

System 6 Chooser:

The 6.0 Chooser's LookupName call to find AppleTalk entities is made asynchronously where retBuffPtr points to a 512-byte buffer and maxToGet = 32 (this is the 32-device limit per device type you may have heard of). The important thing to note here is the return buffer size (512 bytes). For example, if you are looking for AppleShare servers in your own zone (the "*" zone), the number of overhead bytes per NBP tuple returned will be 16 (5 for the entity address, 10 for the string "AFPServer", and 2 for the string "*"). If there were 20 servers in your zone, 340 bytes of the 512-byte buffer are used *before* you start counting the space used by the server names. That leaves 172 bytes for the names or around 8 characters per name (1 length byte plus 8 characters). If the average server name is longer than that, there won't be enough room to collect all of the NBP replies and one or more servers won't show up in the list.

Once a server is selected and the user is authenticated, the AppleShare 2.0 RDEV uses afpGetSrvrParms to ask for the list of server volumes. The AppleShare 2.0 RDEV uses a 512-byte buffer for the replies. After the overhead used by the AppleTalk protocol headers, that's enough room for around 16 volumes with full-sized names; more if the names aren't full-sized.

System 7 Chooser:

The System 7.0 Chooser fixes the problem with the NBP buffer completely. It dynamically sizes the NBP `LookupName` return buffer. So, if `numGotten` => `maxToGet`, it will make the return buffer larger and increase the value of `maxToGet`. The System 7.0 Chooser starts with `retBuffSize`=1024 and `maxToGet`=256.

The AppleShare 7.0 and 3.0 RDEVs increased the size of the `afpGetSrvrParms` reply buffer to 1728 bytes. That's still not big enough to get 255 volumes (the AFP limit) with full-sized names. However, it is big enough for 50 volumes with full-sized names, the maximum number of volumes supported by AppleShare 3.0.

For both System 6 and 7 Choosers, the only limits imposed on the zone are the List Manager limits of 32K of data per list. If each zone name were 33 characters, for example, that would give you space for roughly 1000 zones.

Server and workstation clock times

Date Written: 2/25/92

Last reviewed: 4/22/92

If I hook up two Macintosh computers over LocalTalk, turn on Personal File Share, mount one computer's volume on the other, and make changes to files on each machine, the Get Info mod dates are not adjusted. In my case, machine A's clock said 10:20 and machine B's said 10:30. From machine B, I made a change to a file on machine A. Then, still on machine B, I did a Get Info on that machine A file. Its mod date said 10:20. I then instantly made a change to a file (from machine B still) on machine B and did a Get Info on it and its mod date was 10:30. In other words, the mod dates were not adjusted and reflected the time of the machine each file was located on. Am I misinterpreting something?

The way the workstation computes the server time is not quite as straightforward as is documented in *Inside AppleTalk,* 2nd edition, page 13-21. When a workstation logs onto a server (File Share or AppleShare), the difference between the workstation's clock and the server's clock (s - w) is computed. All subsequent server date/time values as seen by the workstation are computed by adding this difference (s - w) to the server data/time (workstation time = server time + (s - w)).

However, it looks as if the Macintosh workstation also uses the following algorithm to compute the adjusted server time:

- if the offset (s - w) is 15 minutes or less, report the server time as is
- if the offset (s - w) is greater than 15 minutes, compute the offset rounded up to the nearest 30 minute interval.

For example, let's say you have two machines, A and B. B logs on to A. B then goes and modifies a file on A. Listed below are the clock times that the modification took place, and in the rightmost column is the mod. time that B would see for the modified file on A.

```
    A time               B time              mod. time as seen by B
    ------               ------              ----------------------
    11:09                11:22                        11:09
    11:04                11:20                        11:34
     4:34                 7:14                         7:05
```

In the first example, the difference (s - w) is (11:22 - 11:09) = 13 minutes. Since 13 is less than 15 minutes, B sees the server time as is. In the second example, (s - w) is (11:20 - 11:04) = 16 minutes which is greater than 15 minutes so compute the offset to the nearest 30 minute interval (30 - 16) = 14, and 11:20 + 14 = 11:34. In the third example, (s - w) is (7:14 - 4:34) = 2:40 which is greater than 15 minutes so compute the offset to the nearest 30 minute interval (30-40) = -10, and 7:14 - 10 = 7:05. In the last example, (s - w) is (11:53 - 11:22) = 31 minutes so compute the offset to the nearest 30 minute interval 30 - 31 = -1, and 11:53 - 1 = 11:52.

You are probably asking why the 15-minute cushion and why round to the nearest 30-minute interval? Possibly it's an attempt to approximate a modification time somewhere in between the workstation and server times.

PBCatSearch on AppleShare volumes

Date Written: 3/11/92

Last reviewed: 5/21/92

`PBCatSearch` acts differently on a local hard disk than on an AppleShare volume. Say, after a couple of successful `PBCatSearch` operations on a volume, the user modifies the directory by duplicating, renaming, or removing a file. An error is returned, and (theoretically), the search can continue. If you do this on a local hard disk, everything is cool. After making the change, the next `PBCatSearch` call returns -1304 (`catChangedErr`), but subsequent calls return `noErr` and continue to find files. However, if you run this on an AppleShare 3.0 volume, the first call after the change returns -5037 (`afpCatalogChanged`), but all following calls continue to return that error, and "find" the same file that was found on the last good attempt. So what gives?

The `afpCatSearch` AFP call does not map exactly to the File Manager's `PBCatSearch` call. This isn't uncommon in File Manager to AFP translations because AFP calls are designed to be more general so they can be implemented on platforms other than the Macintosh. In some cases, AFP keeps more information than the Macintosh requires (for example, ProDOS file type mapping information for Apple II systems and short names for DOS workstations) and in other cases Macintosh-specific information is "generated" by the Macintosh workstation software (for example, allocation block sizes in the Volume Control Block). Here are specific differences I've found between `PBCatSearch` and `afpCatSearch`:

- afpCatSearch and the AppleShare workstation implementation of `PBCatSearch` do not use `ioSearchTime`. The AppleShare 3.0 server searches for up to 1 second or 4 matches maximum and then returns to the workstation with whatever matches (0-4) are found within areas of the disk that user has access to. The AppleShare workstation keeps asking for the number of matches requested minus the total matches returned until it gets the number requested, or the server returns an error.
- AFP 2.1 does not support both physical and logical fork lengths. If a `PBCatSearch` call uses fork lengths, the upper bound (in the `afpCatSearch` Spec2 field which comes from the `ioSearchInfo2` record) becomes the maximum of the logical and physical lengths and the lower bound (in the `afpCatSearch` Spec1 field, which comes from the `ioSearchInfo1` record) becomes the minimum of the logical and physical lengths.
- AFP 2.1 does not support the `fsSBNegate ioSearchBits` bit. If a `PBCatSearch` call uses `fsSBNegate`, that bit will be ignored by the AppleShare workstation and server and you'll get back exactly the opposite of what you expected. This is an unfortunate omission from AFP 2.1. Because it is implemented this way in at least two shipping servers, the `fsSBNegate` cannot be added without a revision to the AFP specification.
- The File Manager `PBCatSearch` call doesn't return any matches when a `catChangedErr` occurs. However, it does return an updated `ioCatPosition` record which can be used to make another `PBCatSearch` call (this may result in your search either missing a few entries or getting a few duplicate matches). `afpCatSearch` does not work that way. `afpCatSearch` only returns AFP reply data (which includes the ioCatPosition record) if the `FPError` is `noErr` or `afpEofError`. The current `ioCatPosition` record is not returned to the workstation if any other error occurs. So, if an `afpCatalogChanged` error occurs, the `ioCatPosition` record is not returned to the workstation and the workstation returns ioCatPosition to the caller of `PBCatSearch` unchanged. Since the `ioCatPosition` record is still invalid, calling `PBCatSearch` again with the same invalid `ioCatPosition` record will just return the `afpCatalogChanged` error again. The conclusion from this explanation is that you can continue a search if `PBCatSearch` returns a result of noErr or `catChangedErr`. The search completed if `PBCatSearch` returns a result of eofErr. All other results from PBCatSearch (including `afpCatalogChanged`) indicate that you must restart the search from the beginning by clearing the initialize field of the `ioCatPosition` record.

AppleShare Prep file and boot-mounting volumes

Date Written: 8/25/92

Last reviewed: 9/15/92

I have selected AppleShare volumes to mount at system startup by checking the volumes in the Chooser list. If I'm on a nonextended network and I call an extended network via AppleTalk Remote Access and log into a remote server via the Chooser and AppleShare, an error alert will say "The AppleShare Prep file needed some minor repairs. Some AppleShare startup information may be lost" and all the information about my local nonextended network will be cleared out of the AppleShare Prep file, so I loose all my log-in IDs and passwords for my local servers. The same thing happens going back the other way (extended to nonextended). Why is this happening?

There are several problems you can run into when you connect two networks (and that's what you're doing when you use AppleTalk Remote Access when you're already connected to a network). The problems are usually the result of duplicate names or duplicate node numbers.

The "boot mount list" (BML) kept in the AppleShare Prep file stores the location of volumes that you want mounted at boot time. Part of that location is the zone name. If you create entries to the BML when you aren't on an extended network (that is, when you have no zones), the zone name stored in the BML is "*" ("*" is AppleTalk's shorthand for "this zone"). If you create entries to the BML when you are on an extended network (that is, when you have zones), then the zone name stored in the BML is the zone name of the server.

The boot mount code checks the validity of the BML when the system starts up, and the Choose checks the validity of the BML when it's opened. If there are no zones, then entries with zone names other than "*" are cleared out and an alert saying "The AppleShare Prep file needed some minor repairs. Some AppleShare startup information may be lost" is displayed because those entries aren't valid. If there are zones, then entries with zone names of "*" are cleared out and the alert is displayed because the "*" zone name isn't a reliable way to save the zone location of a server on an extended network. The "*" zone isn't reliable for storing the zone name because a workstation can easily be moved from zone to zone, keeping the same NBP object and NBP type names. This is especially true with AppleTalk phase 2, which supports multiple zones on a single network (for example, multiple zones on the same piece of Ethernet cable).

The workaround for boot-mounting volumes is to create alias files to the file servers you want to mount at boot time and then drop those alias files into the Startup folder inside your System Folder. The only drawback to this is aliases don't save the user's password. If you need boot-mounted volumes without the password dialog, you'll have to use guest access.

Back to top

Downloadables

	Acrobat version of this Note (K)	Download

Back to top