Apple®

AppleShare® PC Developer's Guide

Apple Confidential

Final Draft 7/15/88

Contents

Figures and tables

Preface

This manual provides guidelines for writing network-aware applications for personal computers running the MS-DOS operating system (hereinafter referred to as "PCs"). The AppleShare PC network environment is assumed, but many of the concepts are applicable to any multi-processing environment, including other PC network systems and PC multi-tasking systems. AppleShare PC specific concepts, such as directory access privileges and file extension mappings, are covered in separate sections. This manual is valuable reading for all developers interested in writing or updating their existing PC applications to exploit the potential of a multi-user networked environment, including the ability to transparently share data with Macintosh systems.

What you need to know

To use this manual effectively, you should already understand the MS-DOS file system, and how to create and use files and directories from within the programs you write. Familiarity with the C language and the 8086 instruction set are useful, as several program examples are given in these languages. This manual assumes that you have already set up AppleShare PC and are familiar with its use. You do not need to be a Macintosh programmer to use this manual, although some familiarity with the Macintosh would be useful for understanding AppleShare PC's facilities for PC/Macintosh data sharing.

Related documents

Here is a list of some additional books that you may find useful, and who publishes them. (Note: APDA is the Apple Programmer's and Developer's Association.) The last two are only useful if you are developing AppleTalk network programs; they do not directly pertain to AppleShare PC.

- *AppleShare PC User's Guide* Apple Computer, Inc.

  Accompanies AppleShare PC. Describes how to install and use. Package also includes the AppleShare PC Update, which describes the transparent printing facility.

- *DOS 3.3 Technical Reference* IBM (part no. 62800059)

  Authoritative reference for all the DOS function calls.

- *Software Applications in a Shared Environment* APDA

  Similar in scope to this document; provides guidelines for Macintosh programmers writing applications in today's network-shared and multi-tasking world.

- *PostScript Language Reference Manual* Addison-Wesley

  Authoritative reference for PostScript. Useful for programmers who want to add PostScript support to their programs.

- *AppleTalk Filing Protocol (AFP) Engineering Technical Notes* APDA

  Authoritative reference for AFP, the filing protocol used between AppleShare servers and workstations.

- *AppleTalk PC Card and Driver Preliminary Note* Apple Computer, Inc.

  Describes how to use the AppleTalk PC driver to communicate over an AppleTalk network. May be obtained through Developer Services, until published through APDA.

- *Inside AppleTalk* APDA

  Authoritative reference for the AppleTalk network protocols.

Organization

The manual is structured as follows:

- □ Chapter 1 is a brief overview of AppleShare PC.
- □ Chapter 2 discusses AppleShare's privacy features and how they appear to MS-DOS programs.
- □ Chapter 3 discusses the general requirements for being "network aware."
- □ Chapter 4 explains the AppleShare server's sorted directories and their impact on programmers.
- □ Chapter 5 describes how to use several enhancements to the DOS environment that AppleShare PC provides.
- □ Chapter 6 talks about sharing data between PC and Macintosh systems.
- □ Chapter 7 discusses the abilities and limitations of the printing facility.
- □ Appendix A provides programming examples.
- □ Appendix B discusses network and multi-user errors.

You can skip chapter 3 if you are an experienced author of multi-user programs and only need to find out what additional features / constraints are imposed by AppleShare PC.

Chapter 1

AppleShare PC Overview

AppleShare PC is a software package that gives IBM PC or compatible computers transparent access to AppleTalk file and print services. Figure 1-1 shows a simple example of file sharing with AppleShare PC.

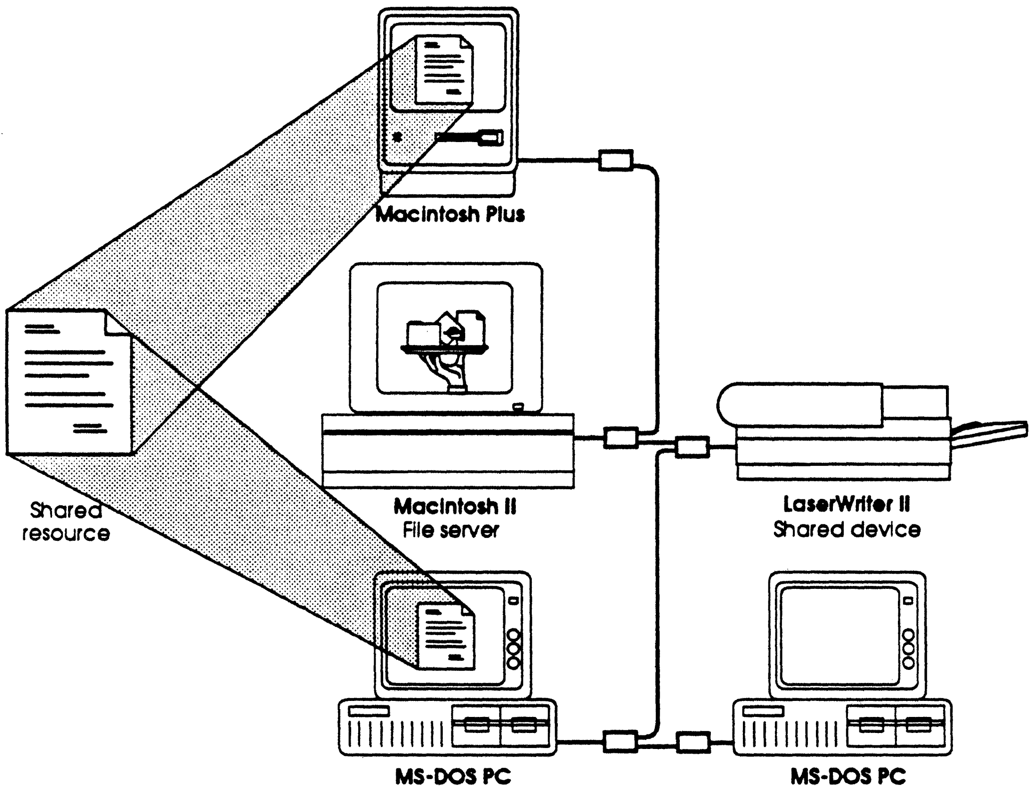

Figure 1-1
File sharing with AppleShare PC

The file service is based on the Microsoft Networks model, but includes an additional module for translating between the SMB protocol spoken by the DOS redirector and the AFP protocol spoken by the AppleShare file server. All elements of the DOS file system are supported by AFP, plus several additional features, such as directory access privileges. Compatibility is at the DOS int 0x21 level – SMB and NetBIOS protocols are not supported. The print facility allows any LPT device to be redirected to a network printer (Apple LaserWriter, ImageWriter II, or ImageWriter LQ). It offers two modes of operation: untranslated, for PostScript (and ImageWriter) savvy applications; and Epson LQ2500 emulation, the de facto printing standard in the PC world.

Two user interface programs are provided. The first is the AppleTalk PC Desk Accessory, or simply DA. It is a menu driven pop-up that is used for locating and connecting to network services, viewing and modifying directory access privileges, entering file and directory comments, plus other network and DOS utility functions. When possible, related windows are modeled after their Macintosh counterparts (the Chooser and Access Privileges desk accessories, the Finder's Get Info window). DA features include on-line help, display of asynchronous notifications from servers, and the ability to save connection information to a configuration file that is read at boot time.

The second user interface program is ANET, the AppleTalk PC Command Line Interpreter. It provides a subset of DA's functions in a command line format, and is useful for constructing batch files to automate certain network-related tasks.

Requirements

AppleShare PC requires an IBM PC or compatible with at least 384k of RAM running DOS version 3.1 or higher. It also requires Apple's AppleTalk PC driver, which provides AppleTalk protocol support, and a driver-supported network card, such as Apple's LocalTalk PC Card.

Chapter 2

Access Privileges

AFP defines a rich set of directory-level access controls that allow users to finely control who can access their information and in what manner. The access restrictions imposed by the server pose a new concern for PC programmers, as prior to AppleShare PC there were no access control mechanisms in MS-DOS, except for the read-only attribute bit for files. In this chapter we will review the AFP access rights information, and then show how privilege violations appear to DOS programs and how to properly support that environment.

How access privileges work

To AFP, every directory has an owner. Initially, a directory's owner is the user who created it, or the special designation *<Any User>*, if the directory was created by a guest. A directory can also be associated with a *group*, a named set of users set up by the AppleShare administrator. Every directory has associated with it three privilege sets, one for the owner, one for the group members, and one for everybody. If a user falls in more than one of the three categories (owner and everybody, for example) his/her effective privileges will be the maximum of the privileges given to all the categories to which he/she belongs.

There are three distinct privileges that can be assigned to each user category. In the AppleShare user documentation they are referred to as See Files, See Directories (See Folders for Macintosh users), and Make Changes. In the AFP specification they are known as Read, Search, and Write, respectively. To perform an operation on an object (file or directory), you usually must have the See Directories privilege for every directory along the path to that object, from the root to the grandparent of the object. In other words, you must be able to search along the path from the root directory all the way to the directory containing the object. Then, you must have sufficient access privileges to the parent directory, which naturally depends upon the operation. For instance, opening a file for read requires Read (See Files) access, deleting a file requires Write (Make Changes) access, and renaming a directory requires both Write (Make Changes) and Search (See Directories). Refer to the AFP documentation for the complete list of file operations and their required access rights. Do not rely upon the "Network Access Rights" discussions in the *DOS Technical Reference*; they do not refer to the AFP privilege system.

How privilege violations appear to DOS programs

In the interests of application transparency, no new error mechanisms were added to DOS. Instead, privilege violations are mapped into existing DOS errors and critical errors. Specifically, all DOS calls that fail due to a privilege violation return an *access denied* error, AX = 5. The only exception is the Find First Matching File call, AH = 0x4E. If you attempt to find files and/or subdirectories in a directory that you do not have the required privileges to, you will receive a *no more files* error, AX = 18. The FCB (File Control Block) version of Find First, AH = 0x11, returns 0xFF in AL to indicate no match. In other words, programs can't tell the difference between an empty directory and one that they don't have the privileges to see into (although they can obtain this information; see the "DOS Enhancements" section).

In the area of file i/o, privilege errors occur at the time the file is opened. This means that you should not normally see an access denied error if you try to read or write a file which you had opened in the appropriate mode. There is one exception, however. The various create calls (Create File, Create Unique File, Create Temporary File) are all defined to create the file and then open it in compatibility mode for read and write access. This poses a problem if the directory where the file is created is a drop box. (A "drop box" for files is a directory for which you have the Make Changes privilege but not the See Files privilege. This allows you to create files within the directory, but not to read or see those files, or reopen them once they are closed.) Drop boxes are very useful, but are unusable to DOS programs if the DOS specification is followed strictly, since any create call will fail for lack of read access. In order to allow files to be created in a drop box, the DOS create call specification has been relaxed. If the newly created file can not be opened for read access, then it is returned opened for write access only. It is therefore possible to receive an access denied error upon the first attempt to read from a file opened via a create call.

❖ *Note:* Since temporary files are ultimately renamed or deleted, and since those operations aren't allowed in a drop box, it is senseless to create temporary files in a drop box. AppleShare PC versions 1.1 and higher will fail the Create Temporary File call (with an access denied error) if you do not have both See Files and Make Changes access to the specified directory. Therefore, this exception does not apply to the Create Temporary File call.

How to work with access privileges

Without AppleShare PC, the only access violation programmers had to handle was trying to write to or delete a read-only file. Now there are many more situations where an access denied error can occur. The fundamental axiom to keep in mind in order to properly operate in AppleShare's privilege environment is **don't make assumptions**. Unlike DOS, AppleShare PC's privileges are non-hierarchical. Being able to write to a file does not imply the ability to read it, being able to make changes to a subdirectory implies nothing about the parent directory or any child directories, being able to create a file or directory does not imply that they will be visible in a directory listing. About the only thing you can be sure of is that if you successfully open a file, you will be allowed to access the file in the manner specified by the open mode (except for the drop box exception explained earlier). Below are some of the consequences of the AFP privilege system, and how you should write your applications to deal with them.

The current directory might not be readable or writable.

Since Search is decoupled from Read or Write, it is possible for users to CHDIR into a directory to which they have no read or write access. To them, it appears to be an empty directory. Therefore, you should not rely upon being able to create files in the current directory. If your application needs to create temporary files, we suggest that you use the directory specified by the TEMP environment variable, if it exists. This is the same environment variable used by Microsoft Windows applications to create their temporary files, and is usually set up by the user to point to a RAM disk or other fast storage device. If the Create Temporary File call fails with an access denied error, you can assume that you do not have sufficient access privileges to the specified directory, and can take appropriate action (ask the user for a different directory name, etc.).

Saving files

Not all the methods currently used for saving files will work properly. For example, creating a temp file and then renaming it fails in a drop box; checking if a file already exists by making a Find First Matching File or Change File Mode call may tell you the file is not there even though it is, or fail with an access denied error even though you are allowed to create the file. The file extension mapping section in chapter six contains an algorithm for saving files that works correctly in the AFP privilege environment.

Chapter 3

Network Awareness & Multi-User Considerations

In this chapter, we will discuss the issues that arise in any multi-access environment, be it a network or a multi-tasking operating system. In general, there are three classes of application program: ***network dumb, network aware,*** and ***multi-user.***

- ☐ *Network dumb* applications were not written for the multi-access environment; they and their data files can reside on a server, but they can only be run by one user at a time. Network dumb applications ignore the problem of multiple access to data files, resulting in possible corruption of data and/or user confusion.
- ☐ *Network aware* applications allow several users to run the application simultaneously, but do not allow simultaneous access to shared data files. Unlike network dumb applications, however, they actively prevent simultaneous access, and therefore ensure data integrity.
- ☐ *Multi-user* applications allow several users to simultaneously access and modify shared data files. An example would be a point-of-sale inventory system, where several sales terminals all simultaneously query and update a central inventory database.

As networks and multi-tasking become prevalent in today's computing environments, network awareness is fast becoming a minimum requirement for applications. Many developers are adopting a two-tiered approach, offering a single-user, network aware product, and a full-scaled multi-user version. With proper design, both versions can share substantially the same source code. This chapter discusses the requirements for being network aware and multi-user.

Atomicity

To be successful in migrating from the single-process environment to that of multiple users and processes, developers must adopt a mind-set of caution and vigilance against race conditions. This is especially true with the advent of large internetworks. High network traffic can flood internetwork gateways and bridges, causing them to drop packets. Retransmission delays are on the order of seconds, so small timing windows between two back-to-back commands can be turned into gaping holes. Appendix A gives an example of extending a file that illustrates the additional work required to ensure proper access.

There's one atomicity-related area where DOS 3.0 proved a great help – creating a new file. You used to have to use the Find First Matching File or Change File Mode calls to check whether the desired file already existed, then proceed with the Create File call if the named file was not found. This method is not reliable over a network, since someone else can create the file just after you've checked to your satisfaction that it's not there. (Also, if you don't have See Files access, these two calls will tell you that the file does not exist even though it might.) To create a file without any chance of overwriting, use the Create New File call (AX = 0x5B). It is implemented atomically by the server as a single command, and will either succeed, fail with a *file exists* error, or fail with an access denied error if you do not have write access to the directory.

Opening files

The most important factor in being network aware is to open files properly. MS-DOS provides a variety of file open modes, which make it possible to control how you access a file, as well as how others can access it. But in order to be effective, a simple, oft-neglected rule must be followed: **Keep data files open while working with them!** Most existing DOS programs open a file, read it into memory, and then close it, only reopening the file later to write any changes back. Any such application is network dumb, because while the memory copy of the file is being worked on, other users/tasks can open the file and make changes to it. At best, those changes will be overwritten without warning when the memory copy of the file is written back to disk; at worst, the data file will become a garbled mix of the two changed versions (if the application uses an incremental approach to saving). Keeping data files open, in conjunction with the appropriate sharing modes, helps ensure data integrity.

Sharing modes

All the DOS open calls take a parameter that indicate how you want to use the file. This parameter is called the *access mode*, and may be either read, write, or read/write. Starting with DOS 3.0, the Open a File call takes an additional parameter which indicates how, if at all, you will allow others to access the file while you have it open. This extra parameter is known as the *sharing mode* or as the *deny mode*. The following four sharing modes are available.

❖ *Note:* In order to use sharing modes, SHARE.EXE, the DOS file sharing support module, must be loaded. Appendix A shows how to check if SHARE is loaded.

Deny Read/Write

Also known as exclusive mode, this completely blocks others from using the file while you have it open.

Deny Write

This mode allows others to read the file while you have it open, but prohibits them from making any changes.

Deny Read

Allows others to make changes to the file while you have it open, but prevents them from reading the file. This mode is not very useful.

Deny None

Allows full read/write access to the file by others. This mode is used by multi-user applications.

As a rule, use the least restrictive sharing and access mode possible. This gives others a chance to access your files, and increases the probability that you will be able to successfully open them. For example, suppose that your program reads a configuration file at start-up. You don't need to write the file, and there's no reason to prohibit others from reading it, so you should open it for read access, deny write. If you request read/write, the open will fail if anyone else has the file open for deny write (or if you do not have the Make Changes access privilege in the file's directory). Similarly, if you request deny read/write, the open will fail if anyone else has the file open for read.

Compatibility mode

There is a special sharing mode used by network dumb applications called *compatibility mode*. Files are in compatibility mode when opened via any of the create function calls, any FCB function call, or if zero is specified in the sharing mode field of the open (as it was in DOS 2.x). Compatibility mode is provided to help old applications function in a network environment.

Compatibility mode files are opened in deny read/write sharing mode, unless the file's read-only bit is set, in which case they are opened in deny write mode. One implication of this is that you can allow a file to be read by multiple network dumb applications if you set its read-only bit. One such application is COMMAND.COM, which uses compatibility mode instead of deny write mode when loading files. This yields an important point to remember: **Even if an application is network aware, you must set its read-only bit to allow several users to load it simultaneously.**

Another characteristic of compatibility mode is that even though the file is opened in deny read/write (or deny write) mode, the opening process is allowed to open the file as many times as it likes. This allows an application to obtain several independent file handles to the same file, which violates the spirit of deny modes and is not especially useful anyway. AppleShare PC supports multiple compatibility mode opens in a limited manner only. The actual access mode for any open after the first will be the minimum of the requested access mode and the access mode of the first open; other access attempts will fail with an access denied error (see Table 3-1).

Table 3-1
Multiple compatibility mode opens

Access mode of first open	Allowed access mode of subsequent opens
Read/Write	Any
Read	Read
Write	Write

Overlays

If you use overlays, you must not deny read access when opening your overlay file; otherwise multiple invocations of your application won't be able to run. If your overlay loader opens files in compatibility mode, you can set the overlay file's read-only bit, which will allow multiple readers. Also remember that network speeds vary, and in general are not as fast as a local hard disk, so you should plan your overlay scheme accordingly.

Byte range locking

Sharing modes are necessary but not sufficient to properly implement a multi-user application; file updates must be coordinated so as to maintain data consistency. The DOS primitive for ensuring atomicity of updates is the *byte range lock*. When you lock a byte range of a file with the Lock File Access call, AX = 0x5C00, you prevent others from reading or writing anywhere within that range of the file until you release the lock. In general, you should lock regions for short periods of time only. DOS automatically attempts access to locked regions several times before returning a lock violation error. You can set the number of retries with the IOCTL Change Sharing Retry Count call, AX = 0x440B.

❖ *Note:* In order to use byte range locking, SHARE.EXE, the DOS file sharing support module, must be loaded. Appendix A shows how to check if SHARE is loaded.

You should use byte range locks whenever you write to a file that does not deny write to others. It is clearly necessary to lock a range while performing a read-modify-write operation, to prevent others from seeing inconsistent data before the write has completed. But something less obvious is that you should byte range lock while performing a write-only operation, such as appending to a data file. This is because writes to a network drive are not atomic; they can be broken up into several small writes. In between these writes, another user can lock part of the range you are trying to write to, or start appending his or her own data, overwriting yours.

Proper use of byte range locking requires careful design, a paranoid programming style, and thorough testing. Incorrect locking schemes are subject to timing-dependent race conditions that may seem to work in a given machine/network configuration; but changing the speed of one of the machines, or the load factor on the network, can bring problems to the surface. A good test method is to embed variable timing loops in your applications, to simulate different CPU and network speeds. The *Multi-User I/O* section of Appendix A gives an example of byte range locking.

Temporary files

Another network-awareness area of caution is the creation of temporary files. Specifically, don't use fixed temporary file names, as another invocation of your application may be running, using the same file name. You can use the "soft" create call (AH = 0x5B) and generate your own unique temporary file names, but we strongly recommend that you use the built-in Create Temporary File call (AH = 0x5A), as this call works correctly in the AFP access privilege environment. Also, as pointed out in the access privileges section, you should try to create your temporary files in the directory pointed to by the TEMP environment variable, if it exists.

Open file categories

The various open deny modes provide for the following four major categories of file access:

- Exclusive (One reader/writer)

  Only one process can have the file open. Useful for application temporary files.
- Browse (Multiple readers)

  Good for shared read-only files, such as help files, configuration files, library files, overlays, and templates.
- Single Writer (One writer, multiple readers)

  A good first step towards multi-user behavior. Allows others to read, while one makes changes. Good for log files, where writing action is append-only.
- Multi-User (Multiple readers/writers)

  Good for databases and other shared centralized data applications.

Table 3-2 lists the access mechanisms required for each of these categories.

Table 3-2
Open file categories

Categories	**Access mode**		**Deny mode**		**Byte range locks required**
	Read	Write	Read	Write	
Exclusive	X	X	X	X	
Browse	X			X	
Single writer (writing to file)	X	X		X	X
Single writer (reading only)	X			X	
Multi-user	X	X			X

By following these guidelines for opening files, byte range locking, and creating temporary files, your application will be network aware, and will operate correctly in multitasking environments. If you then add sensitivity to access privilege violations and follow the guidelines in the chapter on directory enumeration, you will be "AppleShare aware."

Chapter 4

Directory Enumeration

Alphabetically ordered directories

One feature of the AppleShare server is that directories are always maintained in alphabetical order. This is in sharp contrast with MS-DOS, which uses a creation order directory storage mechanism. Files are placed in DOS directories in the order of their creation. If a file is deleted, this leaves a "hole" and the next file that is created will fill that hole. The following example illustrates the difference:

```
C:\> dir

 Volume in drive C is NORMAL DOS
 Directory of  C:\

XYZ                  3   2-12-88
4:55p
        1 File(s)    246272 bytes
free

C:\> copy xyz a
        1 File(s) copied

C:\> del xyz

C:\> copy a c
        1 File(s) copied

C:\> copy a b
        1 File(s) copied

C:\> dir

 Volume in drive C is NORMAL DOS
 Directory of  C:\

C                    3   2-12-88
4:55p
A                    3   2-12-88
4:55p
B                    3   2-12-88
4:55p
        3 File(s)    245248 bytes
free
```

```
D:\> dir

 Volume in drive D is AppleShare
 Directory of  D:\

XYZ                  3   2-12-88
4:55p
        1 File(s)  11131904 bytes
free

D:\> copy xyz a
        1 File(s) copied

D:\> del xyz

D:\> copy a c
        1 File(s) copied

D:\> copy a b
        1 File(s) copied

D:\> dir

 Volume in drive D is AppleShare
 Directory of  D:\

A                    3   2-12-88
4:55p
B                    3   2-12-88
4:55p
C                    3   2-12-88
4:55p
        3 File(s)  11130880 bytes
free
```

Figure 4-1
DOS directory order versus AppleShare sorted directory order

PC programmers never had to think about how DOS maintained its directories, but the truth is that the DOS Find First/Find Next (or *enumeration*) mechanism relies upon the assumption of a creation order storage. Naively implementing the enumeration mechanism on top of AppleShare's sorted directory structure would result in operations such as wild card delete missing files – in fact, DEL *.* would end up deleting every other file.

AppleShare PC presents the illusion of a creation-order directory structure to DOS, so that most wild card directory operations perform as expected. So long as a few simple rules are followed, directory enumeration should look no different to a program than it does on a DOS drive. Let's review the basics of enumerating a directory; then we'll describe the additional rules imposed by AppleShare PC.

Enumeration guidelines

Directory enumeration is done via the Find First Matching File (AH = 0x4E) and Find Next Matching File (AH = 0x4F) calls (there are also FCB equivalents of these calls; they work similarly). To start an enumeration, you call Find First, passing a wild card file specification. DOS returns at the current disk transfer address (DTA) a buffer containing information on the first matching file found, plus a reserved area used by Find Next. To find subsequent files, you call Find Next, which finds the next matching file (using information stored in the reserved area), and updates the reserved area. This continues until there are no more matching files, at which time DOS returns a *no more files* error, AX = 18.

It should go without saying that you should not change or rely upon the contents of the reserved area. For one thing, it is different for AppleShare drives than for normal DOS drives. The DOS program TREE modifies the reserved area instead of using the published DOS interfaces. As a result, TREE does not work on AppleShare drives.

Enumeration lifetime

An important observation is that the reserved area contains all the information DOS needs to resume a directory enumeration. By saving and restoring the reserved area, you can perform depth-first searches that span an entire subtree. Unfortunately, the reserved area is not large enough to hold all the information AppleShare PC requires to resume a directory search. As a consequence, AppleShare PC must allocate memory to save the additional information it needs. With DOS drives, you can save the reserved area to disk, restart the workstation much later, and continue searching where you left off. With AppleShare drives, you can not continue searching once the allocated memory holding the search information has been freed, which occurs in the following situations:

1. After *no more files* has been returned.
2. When the process that issued the Find First exits.
3. If the process that issued the Find First leaves enumerations dangling.

What does it mean to leave enumerations dangling? The best example is a control-c'd DIR command. COMMAND.COM will never complete the interrupted enumeration, in other words, issue Find Nexts until *no more files* is returned. Therefore, the memory associated with that Find First will not be reclaimed by situation 1. To make matters worse, shell programs like COMMAND.COM never exit, so the memory won't be reclaimed by situation 2. If a process performs a Find First, and it already has several (default two) outstanding enumerations within that directory, the process is presumed guilty of leaving enumerations dangling, and all its enumeration data is flushed, meaning that any Find Next based upon a previously saved reserved area will return *no more files.*

Reappearing directory entries

As we mentioned before, AppleShare PC creates the illusion of a creation-order directory structure, so that wild card operations work correctly. However, you must realize the dynamic nature of a directory that is being manipulated simultaneously by several users. Files can come and go while you are enumerating, which gives rise to another rule of enumeration: be prepared to cope with files that seemingly reappear. For example, suppose you are enumerating a DOS directory, and come across file X. Before you are finished enumerating, another application can delete X and re-create it with the net result that it appears a second time during your enumeration (assuming that the recreated X was placed at a later point in the directory which can happen if a third application took the slot of the deleted X; see Figure 4-2).

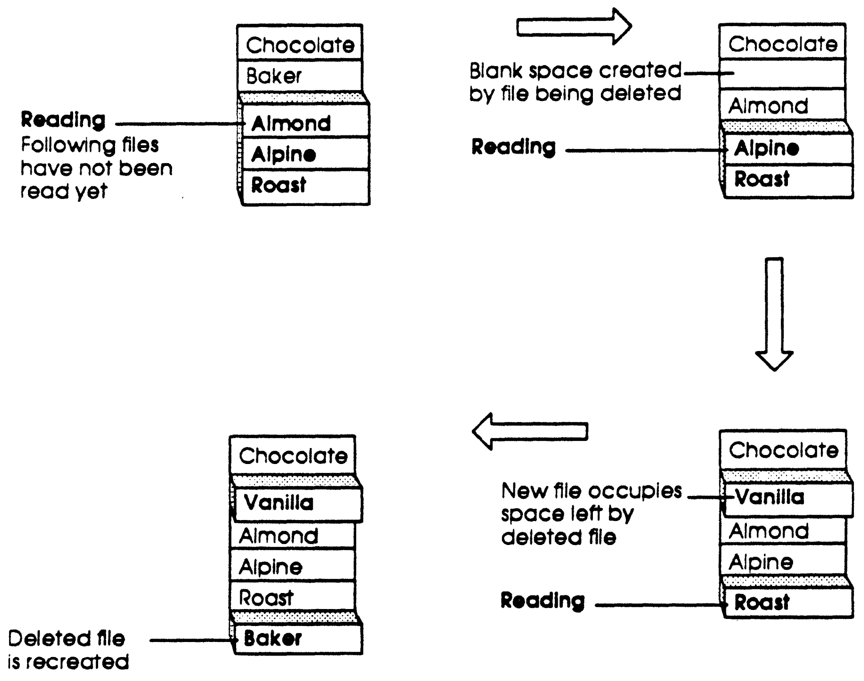

Figure 4-2
Reappearing files under DOS

With AppleShare drives, it's even easier to see X more than once; all that's necessary is for another application to create files in the same directory that are alphabetically less than X (see Figure 4-3).

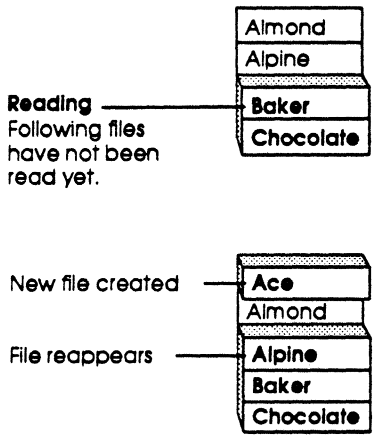

Figure 4-3
Reappearing files under AppleShare PC

If the directory changes enough between Find Nexts, it is possible for the creation-order illusion to break down. We mentioned before that without the illusion of a creation-order directory storage, certain wild card operations can end up missing files. Since it's probably better to see some files twice than to not see some files at all, and since network aware programs must be prepared to filter out reappearing files anyway, AppleShare PC automatically restarts an enumeration whose creation-order broke down. In other words, the next Find Next returns the first matching file, and the enumeration then proceeds normally from that point.

Backing up

There is one more proscribed enumeration activity: "backing up." Backing up means replacing the reserved area with a saved reserved area from the same enumeration. For example, let's say you retrieved 3 matching files, and then replaced the reserved area with the reserved area returned with the first matching file. On a DOS drive, this would cause the next Find Next to "back up" and return the second matching file again. Backing up confuses the creation-order mechanism and produces undetermined results; in the above example, the next Find Next might return the second file, but it might not.

Chapter 5

DOS Enhancements

Although the primary design goals of AppleShare PC were transparency of use and compatibility with existing applications, the following AFP features were made available as DOS enhancements, without sacrificing compatibility or transparency:

- Support of large volumes
- Moving directories
- Enhanced file and directory information

Large volumes

AFP volumes are not limited to 35Mb like normal DOS drives. In fact, as optical storage becomes available, we'll eventually see truly huge volumes. To prepare for the future, make sure that you treat file sizes and disk space as unsigned quantities.

Moving directories

The Rename a File call (AH = 0x56) allows you to rename files and to move files to another directory while renaming. It also allows directories to be renamed, but not moved. AFP has no such restriction, so on AppleShare drives, you can use Rename a File to rename and move a directory tree. For example, you can rename the directory \A\B to \C, which would rename it from B to C, while at the same time moving it from \A into the root directory (see Figure 5-1).

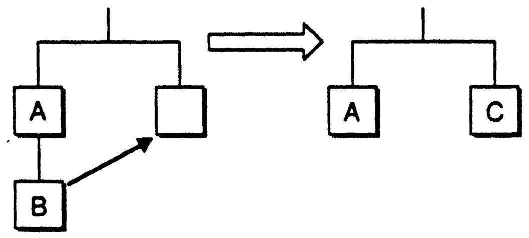

Figure 5-1
Using the Rename a File call to rename and move

Enhanced file and directory information

AppleShare PC can return additional information about files and directories, such as a directory's access privileges. This additional information is returned along with the normal information in response to a Find First/Next call, but only if your application first tells AppleShare PC that it is AppleShare aware. Appendix A describes the special call needed to enable enhanced file and directory information.

Lowercase names

DOS uppercases all file names, so files created or renamed from PCs are always uppercase. Macintosh file names have no such restriction, and files created or renamed from Macintosh computers will most likely be in lowercase or mixed case. Normally, AppleShare PC uppercases all file names before returning them to DOS. When enhanced information is enabled, file names returned will reflect their true case as seen by Macintosh users. This means that AppleShare aware applications must use case-insensitive comparisons for checking file names. Appendix A shows a case-insensitive comparison that includes logic for handling international characters.

Modified bit

Under AFP, each file and directory has three dates associated with it: a creation date, a modification date, and a backup date. The modified bit (IBM calls it an archive bit, but the bit is set when the file has been modified, not when it has been archived) is synthesized by comparing the modification date to the backup date. Since directories have a modification date and backup date too (although they are not accessible from DOS) AppleShare PC can return a meaningful setting of the modified bit for directories. By default, AppleShare PC only sets the directory bit in the attribute word, but it will also set the modified bit properly if enhanced information is enabled. The same applies to the attribute word returned by the Change File Mode call (AH = 0x43). AppleShare aware applications must check whether an object is a file or directory by testing the directory bit (0x10) in the attribute word, not by checking if the attribute word equals 0x10.

Checking access privileges

Normally, the only information applications get about privilege violations is the generic *access denied* error. It would be helpful if applications could check their access to a particular directory before attempting operations. Then they could warn, for example, that writing a file to a drop box is irrevocable; the file can not be changed, deleted, or read back. Programs with a window interface, such as the DA or the Macintosh Finder, use access privilege information to dim or gray inaccessible directories (folders).

The four byte field that returns file size is normally set to zero for directories. When enhanced information is enabled, AppleShare PC returns access privilege information in those four bytes. The format of the four bytes is shown in Figure 5-2 and is described next.

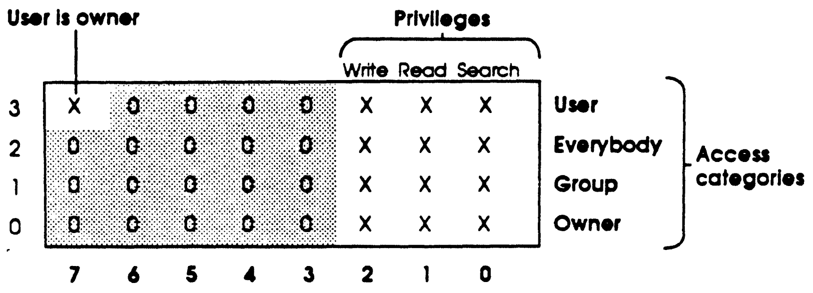

Figure 5-2
Access privileges information field

Byte 3 is typically the byte of interest – it shows the rights granted to the user. If bit 7 is set, then the user is also the directory's owner, and can therefore change the directory's access privileges (with DA or ANET) if necessary. If all four bytes are zero, that would indicate a directory that no one, not even the owner, could access. You can therefore use the same routine for all drives and assume that a zero byte indicates a non-AppleShare drive.

Chapter 6

Integration with Macintosh Applications

Perhaps the most exciting aspect of AppleShare is the potential for heterogeneous computers to directly share data. This chapter discusses building applications that can share data with Macintosh systems, via the AppleShare server.

Common file formats

The ideal data file is one that can be read and manipulated by both PCs and Macintosh Computers without any translation. That way, the file can be used equally by users of either machine, without regard as to which machine created it. So the most important rule for Macintosh/PC integration is to use common file formats. This will allow you to make optimal use of the file extension mapping feature, explained later. Another good idea is to support standard file formats whenever possible, such as PICT format for picture data and WK1 format for spreadsheets, to name only a few. This will allow your application to read a data file without regard as to what application created it.

When designing a common file format, remember to agree upon one representation for numeric and other data. Since the byte ordering of the 680x0 and 80x86 microprocessor families are different, you will probably have to modify or provide your own standard i/o libraries, to ensure that a uniform byte ordering is used in your files.

ASCII text

The lowest common denominator standard format is plain ASCII text. It is very useful to support some form of ASCII input and output, so that text data from other applications can be assimilated. Unfortunately, text representations are not as universal as one might hope. The key difference lies in the choice of end-of-line marker. The Macintosh OS uses carriage return (0xD), A/UX uses line feed (0xA) and MS-DOS uses a carriage return line feed pair. With the advent of AppleShare PC, MS-DOS coprocessors and emulators, and other Macintosh/MS-DOS integration facilities, it would be very desirable to write your i/o routines so that they properly handle any standard end-of-line delimiter combination. The additional work is small, and is well worth the effort. Appendix A includes an example of the C gets ( ) function that accepts any form of delimited text.

A second issue confronting developers that wish to sell internationally is that of foreign character sets. Both IBM and Apple have extended ASCII to 8 bits, using the characters from 128-255 for graphics and foreign characters. Unfortunately, the two extensions are not the same: some characters present in one are not available in the other, and some characters that are available in both extended character sets are represented by different codes.

Although AppleShare PC is a connectivity product, and does not address issues of data content, it does provide support for international characters in file names, which can be used as a model for application developers. AppleShare PC versions 1.1 and higher map all PC file names into the Macintosh extended character set, and perform the inverse mapping on file names returned from enumeration calls. That way, the file names are displayed correctly on both systems, and Macintosh file names also appear normally to DOS users. For consistency with this approach, it is suggested that developers use the Macintosh character set for their common file format, and perform the required conversions in their PC applications. Table 6-1 shows the PC/Macintosh character mapping used by AppleShare PC, along with the lowercase/uppercase mapping used by DOS.

Table 6-1 – PC / Macintosh character mapping

Char	Mapped	Upcased	Char	Mapped	Upcased	Char	Mapped	Upcased
128 Ç	130 Ç		171 ½	219 ¤		214 ╓	227 „	
129 ü	159 ü	154 Ü	172 ¼	221 ›		215 ╫	228 ‰	
130 é	142 é	69 E	173 ¡	193 ¡		216 ╪	229 Â	
131 â	137 â	65 A	174 «	199 «		217 ┘	230 Ê	
132 ä	138 ä	142 Ä	175 »	200 »		218 ┌	231 Á	
133 à	136 à	65 A	176 ░	160 †		219 █	232 Ë	
134 å	140 å	143 Å	177 ▒	164 §		220 ▄	233 È	
135 ç	141 ç	128 Ç	178 ▓	251 ˚		221 ▌	234 Í	
136 ê	144 ê	69 E	179 │	166 ¶		222 ▐	235 Î	
137 ë	145 ë	69 E	180 ┤	168 ®		223 ▀	236 Ï	
138 è	143 è	69 E	181 ╡	169 ©		224 α	237 Ì	
139 ï	149 ï	73 I	182 ╢	170 ™		225 β	167 ß	
140 î	148 î	73 I	183 ╖	171 ´		226 Γ	238 Ó	
141 ì	147 ì	73 I	184 ╕	172 ¨		227 π	185 π	
142 Ä	128 Ä		185 ╣	173 ≠		228 Σ	183 ∑	
143 Å	129 Å		186 ║	175 Ø		229 σ	239 Ô	
144 É	131 É		187 ╗	246 ˆ		230 µ	181 µ	
145 æ	190 æ	146 Æ	188 ╝	247 ˜		231 τ	240	
146 Æ	174 Æ		189 ╜	184 ∏		232 Φ	241 Ò	
147 ô	153 ô	79 O	190 ╛	186 ∫		233 Θ	242 Ú	
148 ö	154 ö	153 Ö	191 ┐	198 ∆		234 Ω	189 Ω	
149 ò	152 ò	79 O	192 └	201 …		235 δ	182 ∂	
150 û	158 û	85 U	193 ┴	202		236 ∞	176 ∞	
151 ù	157 ù	85 U	194 ┬	203 À		237 φ	191 ø	
152 ÿ	216 ÿ	89 Y	195 ├	204 Ã		238 ε	243 Û	
153 Ö	133 Ö		196 ─	205 Õ		239 ∩	244 Ù	
154 Ü	134 Ü		197 ┼	206 Œ		240 ≡	245 ı	
155 ¢	162 ¢		198 ╞	207 œ		241 ±	177 ±	
156 £	163 £		199 ╟	208 –		242 ≥	179 ≥	
157 ¥	180 ¥		200 ╚	209 —		243 ≤	178 ≤	
158 ₧	139 ã		201 ╔	210 “		244 ⌠	248 ¯	
159 ƒ	196 ƒ		202 ╩	211 ”		245 ⌡	249 ˘	
160 á	135 á	65 A	203 ╦	212		246 ÷	214 ÷	
161 í	146 í	73 I	204 ╠	213		247 ≈	197 ≈	
162 ó	151 ó	79 O	205 ═	215 ◊		248 °	161 °	
163 ú	156 ú	85 U	206 ╬	217 Ÿ		249 ∙	250 ˙	
164 ñ	150 ñ	165 Ñ	207 ╧	218 ⁄		250 ·	165 •	
165 Ñ	132 Ñ		208 ╨	220 ‹		251 √	195 √	
166 ª	187 ª		209 ╤	222 ﬁ		252 ⁿ	252 ¸	
167 º	188 º		210 ╥	223 ﬂ		253 ²	253 ˝	
168 ¿	192 ¿		211 ╙	224 ‡		254 ■	254 ˛	
169 ⌐	155 õ		212	225 ·		255	255 ˇ	
170 ¬	194 ¬		213	226 ‚				

File extension mapping

Ideally, if a data file created by a PC is file format compatible with a Macintosh application, Macintosh users should be able to double click on the document icon to run the correct application and work with the file. AppleShare PC has a very powerful *file extension mapping* mechanism. With the appropriate mapping, all data files created by a PC application can be immediately opened by its Macintosh counterpart by double-clicking the icon. Here is how file extension mapping works:

Associated with every Macintosh file are two fields, called the *creator* and *type*. The creator field is a four byte id that tells the Macintosh Finder what application to run if the file's icon is double clicked. The type field is a four byte id that identifies what kind of document the file represents. The type field allows applications to present users with a list of files that they can open, even though those files may not have been created by that application. For instance, a word processor that can read plain ASCII text files in addition to its own files will show in its open dialog box all files with type "TEXT", regardless of their creator. The combination of the creator and type determine what icon will be displayed by the Finder for that file.

AppleShare PC assigns a creator and type to every file it creates, based upon the file's file extension. The default creator/type pair is the creator "mdos" and the type "BINA", which represents a generic MS-DOS binary file. File extensions that are known to represent ASCII files (such as .TXT, .BAT) are assigned the creator "mdos" and the type "crlf", which represents a carriage-return line-feed delimited MS-DOS text file (as opposed to Macintosh text files, which are delimited by carriage returns only). Macintosh applications that want to read and write MS-DOS text files should look for and use the "crlf" file type.

When AppleShare PC mounts a volume, it adds the two icons shown in Figure 6-1 to the volume's desktop database, so that an icon is always displayed for the mdos/BINA and mdos/crlf pairs.

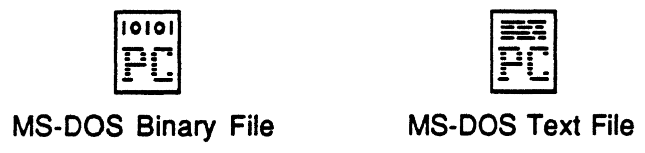

Figure 6-1
Default icons

AppleShare PC is shipped to users with several built-in mappings, and several other creator/type pairs that users can select from. For example, there are several Macintosh programs that can read Lotus 1-2-3 spreadsheets; one of them is set up as the default for the .WK1 file extension, and users can instead select one of the others with DA or ANET. Users never see the actual creators and types; each pair has a name that is used to identify it in DA's Change Extension Mapping menu. For example, the name "DOS-Text" refers to the mdos/crlf creator/type pair.

Adding mappings

If your program produces data files that can be read by a Macintosh application, you should set the appropriate file extension mapping so that your PC-created data files become "double clickable". There are two methods of doing this. One is to add your mapping information dynamically. This is done by making a special call to AppleShare PC, specifying a file extension and creator/type to map to. Appendix A shows how to make this special call. The second method is to add a new named creator/type pair (for example, DOS-Text = mdos/crlf) to DA.DTA, AppleShare PC's configuration file. This adds your pair to the list in DA's Change Extension Mapping menu, allowing users to choose what file extensions map to your pair. You can also add new mapping defaults to DA.DTA, specifying what file extensions map to your new creator/type pair. Table 6-2 contrasts the two methods.

Table 6-2
File extension mappings - adding directly vs. adding to DA.DTA

	Add directly	**Add to DA.DTA**
Mappings take effect	Immediately	Upon reboot
Code supplied by	Developer	Apple Computer
Duration of mapping	Until reboot or until changed	Until changed
User can select from DA/ANET	No	Yes

The format of DA.DTA is proprietary and subject to change. Developers can contact Apple's Developer Services group to receive code that they can incorporate into an installation program that will add new creator/type mappings to DA.DTA. You can also speak to Developer Services about the possibility of pre-configuring your mappings into future releases of AppleShare PC.

Most likely you will only be interested in modifying DA.DTA if you are the supplier of the associated Macintosh application and your mapping information is not already in the DA.DTA shipped with AppleShare PC. In this case, you'll usually perform the modification as part of the installation process. For other developers, it's probably sufficient to set mapping info on the fly. You can send your file extension mapping information to AppleShare PC as part of your application start-up procedure. Another method is to set file extension mappings as part of the saving process. Just prior to saving a file, set the mapping of the file's extension to the desired creator/type. By doing this, all saved files will have the correct creator/type, regardless of their file extension. Of course, since you may be overriding the user's default setting, you should restore the original creator/type after saving. Likewise, if using the start-up method, you should restore the previous creator/type settings when your application exits. Going one step further, you should prompt the user before changing the mapping of any file extension that doesn't map to mdos/BINA or mdos/crlf (the default mappings).

If you are developing a new PC application, you should choose unique file extensions for your data files, if at all possible. That way, your files can be easily distinguished from those of other applications, making the file extension mapping unambiguous.

Saving files

AppleShare PC assigns a creator/type when a file is created and not when it is renamed. This poses a problem for programs that save files by writing a temp file and then renaming the temp file to the destination file name, instead of the easier but less robust method of simply overwriting any existing file. The problem is that the saved files would have an icon based upon the extension of the *temp file*, not the final file name. To remedy this situation, AppleShare PC makes the following special case to handle the above method of file saving:

If a rename call is made where the source is the last file that was created by AppleShare PC and the destination name is the same as the source of the last delete or rename, then the source will be renamed, and its creator/type will be changed to that of the destination file extension.

Given this special case, developers should use the following algorithm for saving files:

```
If the destination does not exist
   write the file to the destination.[1]
else
   write the file to a temporary file
   if backup (.BAK) files are desired
      delete any existing .BAK file
      rename the destination to .BAK[2]
   else
      delete the destination
   endif
   rename the temporary file to the destination[2]
endif
```

Multi-User Reminders:

1 Use DOS function 0x5B (Create New File) so that the test for existence and the subsequent create are an atomic operation.

2 Be prepared for the rename to fail, as another user could create a file of the same target name just prior to your performing this operation.

Chapter 7

Print Access

AppleShare PC provides transparent print access to networked LaserWriters and ImageWriters. Network printers look to applications like an Epson LQ2500 printer attached to one of the LPTn ports; AppleShare PC provides the required translations to the target printer. Users can also disable translation, allowing applications to send PostScript (or ImageWriter) commands directly to printers.

BIOS-level support

AppleShare PC intercepts printer requests at the BIOS (int 0x17) level because many applications use the BIOS for printer i/o instead of DOS. This means that applications using both BIOS and DOS calls to the printer are supported by AppleShare PC. Applications that directly write to the printer ports without going through the BIOS are not supported. In other words, the printer output will not be redirected over the network. Intercepting at the BIOS level has the following ramifications for developers:

- ☐ **BIOS supports more printers than DOS**

  DOS supports only printers designated by LPT1 (or its synonym, PRN), LPT2, and LPT3; the BIOS allows more than three printers (the printer's ID number is just an argument in the DX register). AppleShare PC allows users to specify printer names from LPT1-LPT9; applications that print by means of the BIOS should therefore support these additional printers. In other words, with AppleShare PC's nine printer choices, developers should not assume the absence of an LPT5 just because DOS stops support at LPT3.

- ☐ **Requesting printer status is supported (and is recommended)**

  All the status bits defined for local printers (except the acknowledge bit), are supported on network printers. These bits allow you to determine when the printer is busy, out of paper, and other status situations. Figure 7-1 shows the printer status messages associated with the bit settings.

Printer status messages

Bits	Write fault Print session time out (0x18)	Printer selected (0x11)	Printer out of paper (0x38)	Printer not busy (0x90)	I/O error (0x08)
$08	X		X		X
$01		X			
$10	X	X	X	X	
$20			X		
$80				X	

Figure 7-1
Printer status messages

With PostScript, much of the computation involved with imaging is performed by the printer, not the workstation. Therefore, delays can occur while the printer is drawing a complex graphic, or loading a new font. If you try to send a character to the printer when it is busy, your program will be suspended while AppleShare PC waits for the printer. If you want your program to go on running (in other words, your application performs background printing), you should use the BIOS to request printer status before sending each character. AppleShare PC will clear the high bit of the status byte if the printer is not ready to accept more characters. In this case, do not let your program send a character to the printer. Instead, design your program to wait and retry the print function later (but no later than 4 seconds, as described further on). Also, you should allow a generous period of time for a busy printer to complete a task before reporting an error back to the user. Better still, leave wait-related decisions up to the user by having your program continue trying to print, while providing the user with the option to Cancel.

- ☐ **There is no OPEN command at the BIOS level**

  Network printing requires opening a session with the specified printer. Since the BIOS has no 'open' command, a printer session is established when the first character is sent to the printer. If a session hasn't been established yet, status calls made to a redirected printer port will return that the printer is ready, even though it may not. The only way to check if the printer is truly available is to send the first character of your document. At that time, AppleShare PC will try to open a session with the printer (which can take as long as a minute). If no error is returned from sending the first character, then you can proceed. Otherwise, assume that the printer is temporarily unavailable (it's probably printing another job), and allow the user to retry or abort. If you are printing from DOS, you will receive a critical error if the session could not be opened (for more information, refer to the *DOS-level Support* section later in this chapter).

- ☐ **There is no CLOSE command at the BIOS level**

  Just as the BIOS provides no notice of when to open a printer session, it similarly has no mechanism for an application to tell when it is through with the printer. AppleShare PC uses two mechanisms to determine when an application is done printing: timeout and user notification. There is a "user-settable" timeout (default=5 seconds). If AppleShare PC does not receive a request to print a character or a status request for the printer within this amount of time, it will assume that the print session is over and close the print session. For those applications which print slowly or sporadically (such as the DOS PRINT program), the user can set a much higher timeout value (up to infinity) and manually terminate printing by pressing <ctrl-alt-F10>. It is much better to not require user intervention to close a print session because 1) users shouldn't be bothered with these things, and 2) if they forget to press <ctrl-alt-F10>, the printer can be tied up for a long time, preventing others on the network from using it. Therefore, when printing, you should make sure to send a character or request printer status at least every 3 or 4 seconds, so that you work properly with the normal 5-second session timeout value.

DOS-level support

Printing from DOS is virtually identical to printing from the BIOS, and the same issues apply. This section describes the DOS calls and critical errors for printers.

Printer calls

You can send a character to the printer (usually LPT1) with DOS function 5. The printer is also accessible from the handle i/o calls by means of the handle stdprn = 4. The normal handle i/o calls, specifying either LPT1, LPT2, or LPT3 as the printer name to the OPEN call can also be used as a more generic method of accessing the printer. To request printer status from DOS, use the Get Output Status IOCTL call (AX = 0x4407).

Printer critical errors

The following two DOS critical errors are associated with printing:

□ **Printer not ready (AL = 0x15)**

This error is produced for the user after DOS repeatedly fails to print a character.

❖ *Note*: The actual error signalled is Drive not ready, but this error can be distinguished from a disk-drive problem by the locus value returned by Get Extended Error: for printers the locus will be 4 (serial device) and for disks it will be 2 (block device). Also, the high bit of AH upon entry to the int 0x24 handler will be set for serial devices, and clear for block devices. Since normal printer delays can and do exceed DOS's timeout value, you should retry the operation in response to this error several times before reporting an error to the user. Better still, continue retrying while providing the user with the option to Cancel.

□ **Printer out of paper (AL = 0x1C)**

LaserWriter printers notify AppleShare PC when they are out of paper or the paper trays are removed. (ImageWriter printers do not provide this information on paper status.) The proper response to this error is to provide users with a message indicating that the printer is out of paper and offer an option to Cancel. But design your program to keep retrying the operation until it succeeds or the user intervenes. This will keep AppleShare PC from closing the printer session (due to the inactivity timeout) while the user refills the paper tray.

Epson mode

When in Epson emulation mode, AppleShare PC supports all of the Epson LQ2500 command set, except for color, proportional mode fonts, and downloadable character sets.

❖ *Note:* AppleShare PC does not support the notion of dip switches that can be set by users. If you want a feature or character set normally chosen by a dip switch, send the escape sequence to override the dip switch setting, or AppleShare PC's default, which is undefined, will automatically be selected.

PostScript mode

Most PC applications that support PostScript printing were designed for printers directly connected to the workstation (usually by means of COM1). This led developers to make some design decisions that do not port well to a shared printer environment. Developers writing PostScript applications need to be aware of the following two facts:

□ The printer is not necessarily directly connected through COM1.

This fact presents the following two important guidelines:

1) Allow any LPT port to be specified as a PostScript printer.

2) Do not *a priori* put the printer in hardware-handshaking mode; this does not work with network printers.

□ The printer is a shared resource, within an environment of mixed personal computers.

When several systems make changes to the LaserWriter printer's permanent memory, those changes are likely to conflict. Therefore, do not modify the permanent memory of a shared LaserWriter, except to add a new font. The largest users of LaserWriters are Macintosh workstations. The workstations rely upon the modifications to the LaserWriter printer's permanent memory made by Laser Prep, part of the Macintosh system software.

When global changes are made to the LaserWriter printer's memory (in other words, use the ExitServer construct), the developer risks destroying Laser Prep. If destroyed, the next Macintosh to print to the LaserWriter must reload Laser Prep, a disruptive process. A more serious consequence can occur if the Macintosh does not detect that Laser Prep was reloaded; this causes undefined print problems. Laser Prep is subject to change with each new release of Macintosh system software, so don't assume that your system is functioning properly as long as you don't destroy the current version. A new version of Laser Prep can still be destroyed by your application. AppleShare PC never modifies the permanent memory of the LaserWriter, but it cannot stop a PostScript application from doing so.

Appendix A

Programming Examples

This appendix gives examples of some operations common to most AppleShare aware applications. The examples are shown in Microsoft C, version 4.0, and Microsoft Macro Assembler, version 4.0.

Standard start-up sequence

At start-up, an application should do the following things:

1. Ensure the proper version of DOS is running.
2. Ensure that SHARE is loaded (if using deny modes and byte-range locking).
3. If AppleShare PC is loaded, enable enhanced file and directory information (AppleShare aware applications only).
4. Install a critical error handler for network and printer errors, if desired.

DOS version check

You may not need to check your version of DOS if you require SHARE to be loaded, since SHARE is only present on DOS versions 3.0 and higher. Similarly, if you've determined that AppleShare PC is present, you can assume that you are running DOS version 3.1 or higher. However, if you are building a network-aware application that opens files in compatibility mode only but still want to use the Create Temp File call and any other DOS 3.0 enhancement, you should perform a DOS version check. This is done by making the Get DOS Version Number call (AH = 0x30). From Microsoft C, the version number is already available in the variables _osmajor and _osminor.

Checking for SHARE

The following assembly routine shows how to check for the presence of SHARE.

```
; int share_loaded()
;
; Returns non-zero in AX if SHARE.EXE is loaded; else zero.

_share_loaded   proc
        public  _share_loaded

        mov     ax,1000H        ;SHARE: Get Installed State
        int     2fh
        cmp     al,0ffh         ;AL = FF means SHARE is loaded
        je      xshare
        sub     ax,ax
xshare: ret
_share_loaded   endp
```

Checking for AppleShare PC

If you want to make file extension mapping calls, confirm that AppleShare PC is loaded. The following assembly routine checks if AppleShare is loaded; if loaded, then the routine enables enhanced file and directory information (since the application is presumably AppleShare aware).

```
; int hello_appleshare()
;
; If AppleShare PC is loaded, enables enhanced file and directory
; information and returns the version number of AppleShare PC.
; Otherwise, returns zero.

_hello_appleshare proc
        public  _hello_appleshare

        ; AppleShare PC lives at the int 5c vector.  The four bytes
        ; preceding the vector contain the signature "ASPC", and the
        ; word before that contains the ASHARE version number, with the
        ; major version in the high byte and the minor version in the
        ; low byte.
        mov     ax,355ch        ;get the int 5c vector in ES:BX
        int     21h
        sub     ax,ax
        cmp     es:[bx-4],5341h ;check the signature
        jne     xashare
        cmp     es:[bx-2],4350h
        jne     xashare
        ; AppleShare PC is there.  Enable enhanced file and directory
        ; info for this process.
        push    es:[bx-6]       ;save the version number
        mov     ah,62h
        int     21h
        mov     cx,bx
        sub     bx,bx
        mov     es,bx
        mov     ax,701h
        mov     dl,1
        int     5ch             ;if AX not 0, the call failed (no mem)
        pop     ax              ;get saved version number
xashare: ret
_hello_appleshare       endp
```

Installing a critical error handler

The following assembly routine installs a critical error handler that detects when a file server session has terminated. There are other critical errors that your application will be required to handle; this example demonstrates the general technique.

```
; void install_crit_handler()
;
; Installs an int 24 handler for network critical errors.

oldvec  dd      1 dup(?)        ;storage for the original int 24 vector
srvdead db      13,10,10,"!! File server session has unexpectedly "
        db      "closed, aborting... !!",13,10,10,7,7,"$"

_install_crit_handler   proc    near
        public  _install_crit_handler

        mov     ax,3524h        ;save current int 24 vector
        mov     word ptr oldvec,bx
        mov     word ptr oldvec+2,es

        mov     dx,offset handler ;install our new handler
        push    ds
        push    cs
        pop     ds
        mov     ax,2524h
        int     21h
        pop     ds
        ret
        ; Here's the handler:
handler: push   ax              ;push the world
        push    bx              ;(Get Extended Error nukes all but BP)
        push    cx
        push    dx
        push    si
        push    di
        push    ds
        push    es

        mov     ah,59h          ;call Get Extended Error
        sub     bx,bx
        int     21h

        cmp     ax,55           ;network device no longer exists?
        jne     xhandler        ;nope, chain to normal int 24 handler
        push    cs              ;yes, print fatal error msg
        pop     ds
        mov     dx,offset srvdead
        mov     ah,9
        int     21h
        sub     al,al           ;set zero flag
```

```
xhandler: pop   es                 ;pop the world
          pop   ds
          pop   di
          pop   si
          pop   dx
          pop   cx
          pop   bx
          pop   ax

          jne   chain              ;fatal error?
          mov   al,2               ;yes, tell DOS to abort program
          iret
chain:    jmp   oldvec             ;no, call normal int 24 handler

_install_crit_handler   endp
```

Character handling

This section provides two sample programs (*End of line delimiters* and *International uppercase routine*) that assist developers in building PC programs that can share data directly with Macintosh systems through the AppleShare server.

End-of-line delimiters

This C routine demonstrates how to input a line irrespective of the end-of-line delimiter sequence used (be it *cr*, *lf*, or *crlf*). This allows your application to read text files generated by DOS, Macintosh, and A/UX programs.

```
/* This is a replacement for the standard C gets() function that
 * correctly deals with any cr, lf, crlf, or lfcr line delimiters.
 */

#include <stdio.h>

char *gets(buffer)
   signed char *buffer;  /* buffer to return string in */
{  signed char c;
   register i;

   for (i = 0; ; ++i)
      switch (buffer[i] = getchar())
      {
         case EOF:
            if (i == 0 || ferror(stdin))
               return NULL;   /* error, or EOF and string is null */
            else
               goto got_it;   /* return what we've got so far */
         case '\r': case '\n':
            /* We've read a cr (or lf), if the next char is lf (or cr),
             * eat it.
             */
            if ( (c = getchar()) != (buffer[i] == '\r'? '\n' : '\r') )
               ungetc(c, stdin); /* not an lf (cr), un-eat the char */
         got_it:
            buffer[i] = 0;
            return buffer;
      }
}
```

International uppercase routine

When comparing a file name that a user has typed to a name stored on disk, you must employ a case-insensitive comparison. Likewise, if you have enabled enhanced file and directory information, file names returned from enumeration calls on AppleShare volumes will not be uppercased, and you must be sure to use a case-insignificant comparison when comparing two names. The following assembly routine uppercases a character, using the international uppercase routine supplied by DOS for characters greater than 127. With this routine, you can build an international case-insensitive string comparison function.

```
; toupper(c)
; char c;
;
; This is a replacement for the standard C toupper() function that
; employs the DOS international upcase routine for characters above 7f.

dosupr  dd      getup           ;vector to DOS uppercase routine

_toupper proc
        public  _toupper

c       =       2
        mov     bx,sp
        mov     al,c[bx]        ;(note: assuming SS = DS)
        cmp     al,80h          ;below 80?
        jb      to_up           ;yes, do normal toupper
        call    dosupr
        jmp     short xtoupr
```

```
to_up:  cmp     al,'a'          ;come here for regular chars
        jb      xtoupr
        cmp     al,'z'
        ja      xtoupr
        add     al,'A' - 'a'
xtoupr: sub     ah,ah
        ret
_toupper         endp

; This routine is pointed to by the dosupr vector.  It loads the vector
; with the DOS uppercase routine address and then jumps to it.  With
; this method, no specific initialization routine is needed to load
; dosupr; it's done automatically the first time toupper is called.  If
; your application has a throw-away initialization segment, call toupper
; and then toss this routine.

getup   proc    far
        push    ax              ;save char to upcase
        sub     sp,34           ;get DOS info for current country
        mov     dx,sp           ;(note: assuming SS = DS)
        mov     ax,3800h
        int     21h
        add     sp,18
        pop     word ptr dosupr ;save DOS uppercase routine
        pop     word ptr dosupr+2
        add     sp,12
        pop     ax              ;retrieve char
        jmp     dosupr          ;go upcase it
getup   endp
```

Multi-user I/O

The following example, *Extending a file*, is an example (written in C) that highlights some of the intricacies of multi-user i/o. It demonstrates byte-range locking, and the need to be mindful of race conditions.

Extending a file

```
#include <io.h>
#include <sys/locking.h>/* constants for locking() */
#include <stdio.h>      /* constants for lseek() */


/* This function appends data to a shared data file, and returns the
 * offset to where the data was written, or -1 if the data was not
 * written.
 */
long append(handle, buf, count)
   int handle; /* handle to shared data file */
   char *buf;  /* buffer containing data to be written ;*/
   int count;  /* number of bytes to write */
{  long eof;
   int success;

   for (;;)
   {  eof = lseek(handle, 0, SEEK_END);    /* position to end of file */
      if (locking(handle, LK_NBLCK, count) != 0)
         /* The region is locked, meaning someone is currently
          * appending.  Start over.
          */
         continue;
```

```
        /* The above check is only 95% accurate: given a large network
         * delay, it is possible that another process locked, wrote, and
         * unlocked a new record between the time we seeked to the end of
         * the file and the time we issued our lock.  For the extra 5%,
         * let's try seeking to the end of the file again, and check that
         * the end of file hasn't changed.
         */
        if (lseek(handle, 0, SEEK_END) != eof)
        {  /* The position has changed, i.e. a new record was slipped in
            * while we weren't looking.  Release our lock and start over.
            */
           locking(handle, LK_UNLCK, count);
           continue;
        }
        /* Now we have a lock on the true end of file.  Write the data.
         * If the write is unsuccessful, truncate the file back to the
         * original eof, to remove any partially written data.
         */
        if (!(success = write(handle, buf, count) == count));
           chsize(handle, eof); /* the write failed, reset the file len */
        locking(handle, LK_UNLCK, count); /* release the lock */
        return success? eof : -1L;
    }
}
```

File extension mapping

The following assembly routine allows an application to get and set file extension mappings. AppleShare PC must be loaded to make this call; the first section of this appendix shows how to check if AppleShare PC is loaded.

```
; int mapext(mapbuf, set)
; struct {char ext[4], type[4], creator[4]} *mapbuf;
; int set; /* 1 to set mapping, 0 to get mapping */
;
; mapbuf->ext is a null-terminated file extension (the leading dot is
; not included).  If set is zero, the type and creator for the given
; extension is returned in the mapbuf; if set is one, the type and
; creator are set from the mapbuf.  When setting, mapext returns zero
; for success, or one for failure (out of memory).

_mapext proc
        public  _mapext

mapbuf  =       2
set     =       4
        mov     bx,sp
        mov     dx,mapbuf[bx]   ;(note: assuming SS = DS)
        mov     al,set[bx]      ;(note: assuming SS = DS)
        mov     ah,0
        sub     bx,bx
        mov     es,bx
        int     5ch
        ret
_mapext endp
```

Appendix B

Network and Multi-User Errors

Several additional error codes relating to the network environment were introduced with DOS versions 3.0 and above. Some of these errors are returned from system calls, while others are signalled as int 0x24 critical errors, and can only be obtained by issuing the Get Extended Error call (AH = 0x59) from within the int 0x24 critical error handler. Note that you can not use a dummy critical error handler that fails all critical errors, and then call Get Extended Error after every system call, since the error code returned will be *Fail on INT 24* instead of the actual extended error code. Appendix A shows an example critical error handler. Not all the new errors can occur under AppleShare PC; some of those that can are discussed in this appendix.

Network device no longer exists (AL = 0x37)

This is a critical error that can occur on most any i/o call. It indicates that the file server session has been lost. This usually is a result of network outage, but can also indicate server failure, scheduled server downtime, or a fatal error in the AppleTalk hardware and/or software. Retrying this error is futile.

Unexpected network error (AL = 0x3B)

This is a critical error that is a catchall for many types of NetBIOS errors. For AppleShare PC, it usually is the result of the AppleTalk driver having run out of dynamic memory. This can occur during Find Firsts and file opens, as both these operations allocate memory from the driver memory pool. Although retrying will generally be unsuccessful, an orderly recovery can be attempted, since files already opened can probably still be accessed.

Network data fault (AL = 0x58)

This critical error occurs when the redirector gets a disk full error while trying to flush a buffer. It is different than the normal disk full error mechanism, where the Write Byte Block call (AH = 0x40) returns a number of bytes written that is less than the number requested. *Network data fault* not only indicates that the requested operation has failed, but also that one or more previous write operations, which were now being sent to the server, have failed. Network data faults should be retried several times, since other server users might free up some disk space. If the retries fail, then you must attempt recovery, understanding that some of the data you thought was successfully written to the server actually never made it. To avoid this uncertainty, you can use the Commit File call (DOS 3.3 only, AH = 0x68), or the Disk Reset call (AH = 0xD) to force a file buffer flush, ensuring that all previously written data has been committed to disk. These calls also ensure that the server has flushed any write data from its internal buffers, and thus are a useful checkpointing tool. Network data faults can only occur if you have files open for write access, deny write, because DOS does not buffer writes to files that do not deny write.

Sharing violation (AL = 0x20)

This error is returned when you try to open a file that is already open in a manner that is incompatible with the open file's deny modes. If you are trying to open in compatibility mode, sharing violation is signalled as a critical error; for non-compatibility mode opens, access denied is returned, and sharing violation is returned from the Get Extended Error call. Sharing violation is also returned from Get Extended Error when trying to open a file in non-compatibility mode and the sharing file name buffer has overflowed (see *Sharing buffer overflow* next).

Sharing buffer overflow (AL = 0x24)

The DOS file sharing support module saves the names of all open files in a buffer. If you attempt to open a file in compatibility mode and there is no room for the file name in the buffer, this critical error is signalled. This error is not signalled for non-compatibility mode opens; instead, a Sharing violation error is returned (this is a bug in DOS). Other than reducing the number of open files, the only remedy for this error is to tell the user to increase the /F parameter given to SHARE. Sharing buffer overflow can also be returned from a Lock File Access call; this occurs when SHARE has run out of locks. Here, the remedy is to have the user increase the /L parameter to SHARE.

Lock violation (AL = 0x21)

When an attempt is made to read or write a locked region, access denied is returned, and lock violation is returned from the Get Extended Error call. Lock violation is also returned from the Lock File Access call if you try to lock an already locked region or unlock a region that is not locked or is not locked by you.